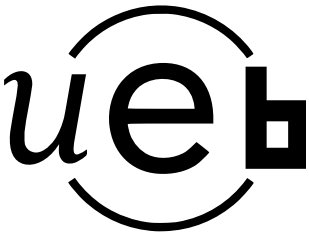
# Représentations parcimonieuses et apprentissage de dictionnaires pour la compression et la classification d'images satellites

Jérémy Aghaei Mazaheri

ANNÉE 2015

**THÈSE / UNIVERSITÉ DE RENNES 1**
*sous le sceau de l'Université Européenne de Bretagne*

pour le grade de

**DOCTEUR DE L'UNIVERSITÉ DE RENNES 1**

*Mention : Traitement du signal et télécommunications*

**École doctorale Matisse**

présentée par


# Jérémy AGHAEI MAZAHERI

préparée au centre de recherche INRIA Rennes - Bretagne Atlantique
Institut National de Recherche en Informatique et en Automatique


# Représentations parcimonieuses et apprentissage de dictionnaires pour la compression et la classification d'images satellites

**Thèse soutenue à Rennes**
**le 20 Juillet 2015**

devant le jury composé de :

**Jean-Yves TOURNERET**
Professeur ENSEEIHT / *Rapporteur*
**Jean-Marie NICOLAS**
Professeur Télécom Paris Tech / *Rapporteur*
**Luce MORIN**
Professeur INSA / *Examinatrice*
**Pierre-Luc GEORGY**
Ingénieur Airbus Defence & Space / *Examinateur*
**Claude LABIT**
Directeur de recherche INRIA / *Directeur de thèse*
**Christine GUILLEMOT**
Directeur de recherche INRIA / *Co-directrice de thèse*

*Quand tout semble inutile, je vais observer un tailleur de pierre à l'œuvre. Il peut marteler une pierre une centaine de fois sans qu'elle ne montre la moindre fissure. Pourtant au cent et unième coup elle se fend soudain en deux, et je comprend que ce n'est pas ce dernier coup qui a produit cet effet mais bien tous ceux qui l'ont précédé.*

Jacob Riis

*A mes parents, pour leur inconditionnel soutien*
*A mon grand frère, qui a toujours été un modèle pour moi*
*A ma petite sœur, pour tous les fous rires qu'on a pu partager*

# Remerciements

J'adresse tout d'abord mes plus sincères remerciements à mes deux directeurs de thèse. Merci à Claude Labit pour ses précieux conseils et pour avoir toujours su me conforter dans la direction à suivre. Merci à Christine Guillemot pour son encadrement, ses remarques et son expérience ont été d'une grande aide dans la réalisation de cette thèse.

Je remercie ensuite mes encadrants au sein d'*Airbus Defence & Space*, Pierre-Luc Georgy et Mathias Ortner, pour leur suivi tout au long de cette thèse. Merci pour les nombreuses données qu'ils m'ont fournies et qui m'ont permis de réaliser ces travaux, ainsi que pour leur expertise dans le domaine de l'imagerie satellite.

Je souhaite remercier tout particulièrement mon jury de thèse : Jean-Yves Tourneret, Jean-Marie Nicolas, Luce Morin et Pierre-Luc Georgy, d'avoir accepté de lire et de juger mon travail de thèse. Merci pour le temps accordé à mon manuscrit et à ma soutenance et pour les remarques apportées.

Je ne peux ensuite pas oublier de remercier les membres, passés et présents, de l'équipe SIROCCO (et précédemment TEMICS) pour leur accueil, les discussions et les nombreuses heures passées ensemble ayant contribué à rendre agréables ces années passées au sein de l'INRIA.

Je remercie tout particulièrement Aline Roumy pour les discussions intéressantes autour des algorithmes de poursuite utilisés au sein du schéma de codage.

Enfin, un immense merci à Elif Vural pour les heures passées à discuter autour du chapitre 6 de classification. Merci pour son écoute, ses conseils et ses contributions, sans lesquels ce dernier chapitre n'aurait pas été le même.

# Table des matières

# Introduction

4 octobre 1957, Spoutnik 1, le premier satellite artificiel de la Terre, est mis en orbite par l'URSS, marquant le début de la course à l'espace avec les États-Unis, plus tard marquée par les premiers pas de Neil Armstrong sur la Lune en 1969. Sphère de 58 cm de diamètre, pesant 83,6 kg et dotée de quatre antennes, Spoutnik 1 émettait un "bip-bip" aujourd'hui devenu célèbre.

Depuis cette date, l'usage des satellites s'est considérablement élargi grâce aux progrès de la technologie. Ces derniers sont en effet actuellement utilisés pour les communications, le guidage GPS ou encore pour la prise d'images de la Terre.

Les images satellites sont ainsi devenues stratégiques pour observer la Terre vue du ciel. Elles peuvent par exemple permettre d'évaluer l'ampleur des dégâts suite à une catastrophe naturelle, de prédire la météo, d'étudier des phénomènes tels que l'urbanisation ou la déforestation, ou encore de surveiller certaines parties du globe.

## Motivations

Les images satellites étant des images de très hautes résolutions, il est nécessaire de les compresser à bord pour pouvoir les envoyer plus rapidement sur Terre, où elles sont alors décompressées avant de pouvoir être utilisées. Le standard de la compression d'images satellite, nommé CCSDS 122 [fSDS05], proche dans son fonctionnement de JPEG 2000 [TM02, SCE01, MGBB00] applique une transformée en ondelettes de l'image. Quant aux standards de compression usuels H.264/AVC [WSBL03] et son successeur HEVC [SOHW12], ils appliquent une transformée proche de la transformée DCT à l'image résiduelle obtenue suite à une étape de prédictions spatiales (et temporelles dans le cas de la vidéo).

Étant donné que nous souhaitons travailler spécifiquement sur des images satellites, nous allons chercher à adapter la transformée à ce type d'images dans le but de la rendre plus efficace. Ainsi, plutôt que d'utiliser des transformées génériques et prédéfinies, nous allons apprendre la transformée grâce à des méthodes d'apprentissage de dictionnaire, utilisées dans le cadre des représentations parcimonieuses.

Les représentations parcimonieuses consistent à approximer un signal, mis sous la forme d'un vecteur, par une combinaison linéaire de quelques colonnes seulement, dites atomes, d'une matrice appelée dictionnaire. Le signal est alors représenté par un vecteur parcimonieux contenant seulement quelques coefficients non nuls. L'apprentissage du dictionnaire sur des données d'entraînement permet de l'adapter à un type de données

particulières et ainsi d'améliorer la qualité de représentation de ces données et de rendre les représentations plus parcimonieuses.

Utiliser les représentations parcimonieuses dans le cadre du codage nécessite de coder ces coefficients non nuls ainsi que les indices des atomes du dictionnaire auxquels ils correspondent. Ces derniers dépendent directement de la taille du dictionnaire. En effet, augmenter la taille du dictionnaire afin d'améliorer sa capacité de représentation implique également une hausse du coût de codage des indices.

C'est pourquoi des dictionnaires structurés ont été créés [ZGK$^+$10, ZGK11], plus adaptés au problème du codage car pouvant contenir davantage d'atomes sans pour autant augmenter le coût de codage de chaque indice. Ces structures sont constituées de plusieurs dictionnaires organisés en niveaux, chaque dictionnaire étant appris sur des résidus du niveau supérieur. Dans la lignée de ces travaux, nous avons travaillé sur de nouvelles structures de dictionnaires, allant jusqu'à adapter la structure durant l'apprentissage en fonction des données d'entraînement disponibles, afin d'améliorer l'efficacité du dictionnaire.

Nous souhaitons également traiter un second cas d'étude : la classification des images satellites. Nous distinguons alors deux applications distinctes : l'estimation de la Fonction de Transfert de Modulation (FTM) d'un instrument à partir d'une image capturée par ce dernier et la reconnaissance de scènes prédéfinies au sein d'une image satellite, correspondant à un problème de classification supervisée. Ces deux cas distincts de classification traitent des classes de natures différentes. De plus, la première réalise une classification globale de l'image, tandis que la seconde effectue une classification locale en affectant un label à chaque pixel de l'image.

La FTM (Fonction de Transfert de Modulation) est un moyen d'évaluer les performances en matière de résolution spatiale d'un système d'imagerie, c'est-à-dire son niveau de préservation des détails spatiaux. La FTM d'un système d'imagerie optique représente donc un indice de performance du système vis-à-vis de la qualité des images produites. Le système optique à bord d'un satellite pouvant se dégrader suite au décollage ou à des perturbations extérieures, il est utile de pouvoir estimer la FTM du système en orbite pour vérifier son bon fonctionnement.

Les méthodes classiques d'estimation de la FTM utilisent des mires au sol ou cherchent des formes caractéristiques particulières dans les images telles que des ponts ou des bords d'immeubles à fort contraste. Afin d'éviter d'avoir à orienter le satellite sur une zone spéciale, nous décidons d'apprendre des dictionnaires structurés spécifiques à des valeurs de FTM et de les utiliser dans un algorithme d'estimation appliqué à une image de FTM inconnue.

L'application de reconnaissance de scènes consiste à reconnaitre au sein d'une image satellite les différents types de scènes qui la composent, telles que de la mer, de la ville, de la forêt, du désert ou encore des nuages. Cette étude présente des applications concrètes telles que le suivi de l'urbanisation, de la déforestation ou encore de la progression du désert.

Cela correspond à un problème de classification supervisée, cadre dans lequel des dictionnaires appris ont déjà été utilisés. *Mairal et al.* [MBP$^+$08a] proposent par exemple

d'apprendre un dictionnaire par classe tout en les discriminant pour mieux répondre au problème de classification. Dans la lignée de ces travaux, nous allons utiliser les dictionnaires structurés pour leur capacité d'apprentissage vis-à-vis des données d'entraînement, tout en discriminant le premier niveau de chaque structure à l'aide d'une fonction de coût plus simple à optimiser.

## Contributions

Pour ces applications, différentes structures de dictionnaires sont ainsi étudiées. Ces structures permettent de contenir davantage d'atomes qu'un dictionnaire non structuré afin d'améliorer leur capacité de représentation tout en conservant un coût de codage de chaque indice et une complexité de décomposition équivalents. De plus, elles sont scalables en parcimonie dans le sens où elles ne sont pas apprises pour une parcimonie fixée. Nous partons d'une structure arborescente de dictionnaires, comportant de plus en plus de dictionnaires à chaque niveau. Puis, dans une deuxième structure, nommée structure en "cerf-volant", les branches sont élaguées après un certain niveau pour n'apprendre plus qu'un seul dictionnaire par niveau. Enfin, une structure adaptative est étudiée. Dans cette structure, les différentes branches sont progressivement refermées, selon leur popularité vis-à-vis des données d'apprentissage, et fusionnées au sein d'une branche commune. Cette structure s'adapte ainsi automatiquement aux données d'entraînement durant l'apprentissage afin de rester efficace. Les algorithmes de poursuite classiques (MP et OMP) sont également adaptés à ces structures particulières.

Dans le cadre de la compression des images satellites, un schéma de codage basé sur les dictionnaires structurés [ZGK11] est adapté à la Structure Adaptative et quelques améliorations y sont apportées. Ce schéma fonctionne pour un débit cible et sélectionne à chaque étape un bloc sur un critère débit-distorsion pour y ajouter un atome dans sa représentation, c'est-à-dire en codant pour ce bloc un indice d'atome et un coefficient supplémentaires. Les améliorations apportées portent sur le codage des valeurs moyennes des blocs, ainsi que le codage des coefficients, en améliorant l'apprentissage des tables de Huffman et en intégrant la quantification à la décomposition dans le schéma de codage. Le *bitstream* obtenu possède de plus une propriété de scalabilité.

Les dictionnaires structurés appris sont également utilisés dans un algorithme d'estimation de FTM. Une Structure Adaptative est apprise par valeur de FTM. Une image de FTM inconnue est ensuite décomposée sur les différentes structures pour une même parcimonie. La qualité de reconstruction obtenue sur chaque structure permet ensuite, dans un algorithme de décision utilisant des seuils pouvant également être appris, d'estimer la valeur de FTM liée à l'image.

Enfin, dans le cadre de la reconnaissance de scènes, un algorithme de classification supervisée permet de segmenter une image en différents labels. Les dictionnaires structurés utilisés pour cette application sont rendus discriminants lors de l'apprentissage afin d'être davantage adaptés au problème de classification. Ainsi, un dictionnaire structuré est appris par classe, en cherchant à le rendre bon pour sa classe mais également mauvais pour les autres. Pour chaque pixel à classifier, l'erreur de reconstruction sur chaque dictionnaire du patch autour de ce pixel est calculée et utilisée lors d'étapes

de lissage, permettant d'obtenir une segmentation de l'image test propre. Enfin, pour traiter les images satellites en couleurs, un terme de pénalité lié à la couleur est ajouté, sans nécessiter de réapprendre les dictionnaires.

## Organisation de la thèse

La thèse est organisée en trois parties : la première est consacrée à l'état de l'art, la deuxième à l'étude des structures de dictionnaires et leur application au codage d'images satellites et enfin, la troisième partie est consacrée aux applications de classification des images satellites. Chaque partie est constituée de deux chapitres.

Le chapitre 1 est consacré à un état de l'art sur les représentations parcimonieuses et l'apprentissage de dictionnaires. Des méthodes de décomposition, en particulier des algorithmes de poursuite, ainsi que d'apprentissage de dictionnaires y sont détaillées. Des applications des dictionnaires appris aux cas du débruitage, de la compression et de la classification supervisée sont de plus présentées.

Le chapitre 2 dresse un état de l'art de quelques standards de compression d'images fixes : JPEG, JPEG 2000 et le standard consacré aux images satellites CCSDS 122. Les modes Intra des codeurs vidéos H.264/AVC et HEVC sont également décrits.

Dans le chapitre 3, différentes structures de dictionnaires sont étudiées. Nous partons d'une structure arborescente simple, puis celle-ci évolue vers une structure dite en "cerf-volant" et enfin vers une structure adaptative. Ces structures sont comparées à des dictionnaires "plats" appris avec K-SVD [AEB06] ou Sparse K-SVD [RZE10], mais également à d'autres dictionnaires structurés. Des expérimentations sont réalisées avec ou sans le retrait de la valeur moyenne de chaque patch d'apprentissage et de test. De plus, différentes décompositions (de type MP ou OMP) sont appliquées sur les dictionnaires structurés.

Le chapitre 4 propose d'intégrer les dictionnaires au sein d'un codeur simple dans un premier temps, afin de comparer les différentes structures. Puis la Structure Adaptative est intégrée dans un schéma de codage plus optimisé faisant intervenir une optimisation débit-distorsion, de façon à se comparer aux standards de compression d'images fixes. Un aparté est ensuite réalisé au sein de HEVC en version Intra, en y intégrant des dictionnaires structurés appris à l'étape de transformation, en remplacement de la transformation DCT. Le cas de l'observation persistante traitant une séquence quasiment fixe est finalement abordé. Le dictionnaire est alors spécialisé en l'apprenant sur les premières images de la séquence et en testant la dernière image. L'objectif est d'observer les résultats de l'apprentissage dans un cas d'étude quasiment idéal.

Le chapitre 5 présente la méthode d'estimation de FTM. Un état de l'art de quelques méthodes d'estimation est d'abord réalisé. Puis des expérimentations préliminaires permettent de déduire un algorithme de décision basé sur la qualité de représentation d'une image test grâce aux dictionnaires appris pour les différentes valeurs de FTM. Une méthode de détermination des seuils de décision utilisés est également décrite. La méthode d'estimation est enfin testée sur des images satellites pour de nombreuses et proches valeurs de FTM.

Dans le chapitre 6, le problème de classification supervisée pour la reconnaissance

de scènes est traité à l'aide des dictionnaires structurés appris. Un modèle de discrimination est présenté, son but étant de discriminer les dictionnaires des différentes classes à l'apprentissage, puis les méthodes de lissage appliquées, à savoir un lissage par expansion $\alpha$ des labels sur un graph-cut et un lissage par érosion. Les expérimentations sont d'abord menées sur une base connue d'images de textures, afin de comparer notre méthode à la littérature. Puis l'algorithme de classification est appliqué à des images satellites en couleurs. Un terme de pénalité sur la couleur est alors ajouté avant le lissage pour aider la classification.

Enfin, la thèse est conclue par un résumé de nos contributions et des perspectives vers de potentiels futurs travaux sont données.

# Première partie

# État de l'art

# Chapitre 1

# Représentations parcimonieuses et apprentissage de dictionnaire

Transformer un signal, de façon à le représenter dans un autre espace grâce à quelques coefficients seulement, a de quoi séduire. Cette nouvelle représentation du signal est alors dite parcimonieuse, car basée sur quelques coefficients non nuls seulement. Quant à ce nouvel espace de représentation, il est défini par une matrice nommée dictionnaire, dont les colonnes sont communément appelées atomes. On définit ainsi les représentations parcimonieuses comme des représentations de signaux, sous formes de vecteurs, par des combinaisons linéaires de quelques atomes seulement d'un dictionnaire. Les méthodes traitant et utilisant ce type de représentation, cherchant à obtenir une solution la plus parcimonieuse possible, se sont considérablement développées pour faire des représentations parcimonieuses aujourd'hui un sujet de recherche très actif. Elles sont ainsi utilisées dans de nombreux domaines tels que le débruitage, l'inpainting (consistant à remplir la partie manquante d'une image), la super-résolution (permettant d'augmenter la résolution d'une image), la classification ou encore la compression.

Dans ce premier chapitre, nous commencerons par présenter les représentations parcimonieuses ainsi que quelques algorithmes de décomposition parcimonieuse permettant de résoudre ce problème. Nous nous intéresserons ensuite au cas de l'apprentissage de dictionnaire, de façon à apprendre un nouvel espace de représentation adapté aux données pour rendre les décompositions plus parcimonieuses et plus efficaces. Enfin, nous insisterons sur plusieurs applications pratiques des représentations parcimonieuses et de l'apprentissage de dictionnaire, dont certaines seront développées dans le cadre de cette thèse.

## 1.1 Représentations parcimonieuses

Les représentations parcimonieuses consistent à représenter un signal comme une combinaison linéaire de quelques colonnes, dites atomes, d'un dictionnaire, souvent sur-complet. Des algorithmes de décomposition parcimonieuse dits gloutons ont été développés afin de résoudre ce problème en utilisant différentes normes pour forcer la

parcimonie. Récemment, des algorithmes utilisant la notion de parcimonie structurée ont également fait leur apparition.

### 1.1.1 Formulation du problème

Le problème des représentations parcimonieuses est de décomposer un signal $y \in \mathbb{R}^n$ sur un dictionnaire $D \in \mathbb{R}^{n \times K}$ avec une contrainte de parcimonie sur la représentation, c'est-à-dire une contrainte sur le nombre de colonnes de $D$ choisies dans la décomposition (Fig. 1.1). On recherche ainsi en théorie la solution du problème suivant :

$$\min_x ||x||_0, \text{ sous la contrainte } y = Dx \tag{1.1}$$

où $x \in \mathbb{R}^K$ est la représentation parcimonieuse de $y$. $||x||_0$ représente la $norme - l_0$ de $x$ et correspond au nombre de valeurs non nulles de $x$ (ce n'est en réalité pas une norme). Le dictionnaire $D$ est composé de $K$ colonnes $d_k, k = 1, ..., K$, appelées atomes, chacune d'elles supposée normalisée, c'est-à-dire de $norme - l_2$ unitaire : $||d_k||_2 = 1, \forall k = 1, ..., K$. Le dictionnaire est en général sur-complet : il comporte davantage d'atomes que la dimension de chaque atome qu'il contient ($K > n$). Si $K < n$, on dira que le dictionnaire est sous-complet, et complet si $K = n$.

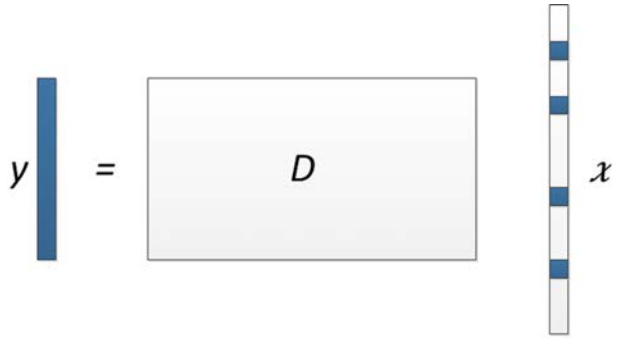



FIGURE 1.1 – Principe des représentations parcimonieuses : $y = Dx$ avec $x$ un vecteur parcimonieux.

Il existe en théorie une infinité de solutions $x$ au problème $y = Dx$ et l'objectif est de trouver la solution la plus parcimonieuse possible, c'est-à-dire celle présentant le plus faible nombre de valeurs non nulles dans $x$. En pratique, on cherche une approximation du signal et le problème devient :

$$\min_x ||x||_0, \text{ sous la contrainte } ||y - Dx||_2 \leq \epsilon \tag{1.2}$$

avec $\epsilon \geq 0$ une erreur admissible et $||.||_2$ la $norme - l_2$ : $||x||_2 = \sqrt{\sum_{i=1}^{K} |x_i|^2}$.

Une autre écriture commune du problème est de chercher à minimiser l'erreur de représentation en fixant une contrainte de parcimonie sur $x$ :

$$\min_x ||y - Dx||_2, \text{ sous la contrainte } ||x||_0 \leq L \tag{1.3}$$

avec $L > 0$ la contrainte de parcimonie, c'est-à-dire un entier représentant le nombre maximal de valeurs non nulles dans $x$.

Résoudre ce problème est NP-difficile, ce qui écarte toute recherche exhaustive de la solution. C'est pourquoi des algorithmes de décomposition parcimonieuse dits gloutons ont vu le jour afin de trouver une approximation de la solution.

### 1.1.2 Algorithmes de décomposition parcimonieuse

#### 1.1.2.1 *Matching Pursuit*

L'algorithme *Matching Pursuit* (MP) [MZ93] est un algorithme de décomposition parcimonieuse dit glouton permettant d'obtenir une approximation sous-optimale du problème (1.3).

Le principe est de sélectionner à chaque itération l'atome du dictionnaire le plus corrélé au résidu courant, mis à jour après la sélection de chaque nouvel atome. Le résidu courant à une itération donnée est défini comme la différence entre le signal original et son approximation courante via les atomes sélectionnés jusque là. L'erreur de reconstruction est ainsi progressivement réduite.

Initialement, le résidu $r^0$ est égal au signal $y$ à décomposer et $x^0 = 0$ (vecteur nul). Puis à chaque itération $i$, le résidu courant est projeté sur chaque atome du dictionnaire, de façon à trouver l'indice de l'atome le plus corrélé au résidu, c'est-à-dire l'indice vérifiant :

$$k^i = \arg\max_k |d_k^T r^{i-1}|, \ k = 1, ..., K. \tag{1.4}$$

Le coefficient correspondant est alors calculé ainsi :

$$\alpha_{k^i} = d_{k^i}^T r^{i-1} \tag{1.5}$$

et intégré dans $x^{i-1}$ pour former $x^i$. Si l'atome a déjà été sélectionné dans la décomposition, alors ce coefficient est ajouté à celui précédemment calculé pour ce même atome. Et le résidu est mis à jour :

$$r^i = r^{i-1} - \alpha_{k^i} d_{k^i}. \tag{1.6}$$

Le critère d'arrêt de l'algorithme peut être une erreur de reconstruction du signal de départ à atteindre, ou une limite de parcimonie sur $x$ correspondant à un nombre maximal d'atomes pouvant être sélectionnés dans la décomposition.

L'avantage de cet algorithme tient dans sa simplicité. Cependant, avec le MP, un atome peut être sélectionné plusieurs fois, ce qui peut ralentir la convergence de l'algorithme vers une solution. C'est pourquoi une variante de l'algorithme *Matching Pursuit*, appelée *Orthogonal Matching Pursuit*, a été créée.

#### 1.1.2.2 *Orthogonal Matching Pursuit*

L' algorithme *Orthogonal Matching Pursuit* (OMP) [PRK93] est basé sur le même principe que l'algorithme MP : sélectionner à chaque itération l'atome le plus corrélé au résidu courant. Mais contrairement à MP, OMP recalcule à chaque itération l'ensemble des coefficients calculés aux itérations précédentes, correspondant aux atomes déjà choisis, de telle sorte que le résidu calculé soit orthogonal à la fois à l'atome juste sélectionné mais également à tous les atomes précédemment sélectionnés. Cette propriété d'orthogonalité rend la projection du résidu sur les atomes déjà sélectionnés nulle. De cette façon, un atome ne peut pas être sélectionné plusieurs fois et l'algorithme converge plus rapidement que MP. Cela permet d'obtenir la meilleure approximation possible en utilisant les atomes sélectionnés jusque là.

Initialement et comme pour l'algorithme MP, le résidu $r^0$ est égal au signal $y$ à représenter et $x^0 = 0$ (vecteur nul). A chaque itération $i$, l'atome du dictionnaire $D$ le plus corrélé au résidu courant est sélectionné de la même façon que pour MP et son indice est donné par :

$$k^i = \arg\max_k |d_k^T r^{i-1}|,\ k = 1, .., K. \tag{1.7}$$

Cet atome est ensuite concaténé au sous-dictionnaire $D^{i-1}$ qui contenait tous les atomes précédemment sélectionnés pour former $D^i = [D^{i-1}, d_{k^i}]$.

On cherche alors la meilleure représentation du signal $y$ sur ce sous dictionnaire $D^i$ en résolvant :

$$\min_{\alpha^i} ||y - D^i \alpha^i||_2^2. \tag{1.8}$$

La solution du problème ci-dessus donne les coefficients correspondant à tous les atomes sélectionnés jusqu'à l'itération $i$ :

$$\alpha^i = ({D^i}^T D^i)^{-1} {D^i}^T y = {D^i}^+ y \tag{1.9}$$

où ${D^i}^+$ est la pseudo-inverse de $D^i$.

Les coefficients ainsi calculés remplacent leur précédente version dans $x^{i-1}$ pour former $x^i$. Et les résidus sont mis à jour comme suit :

$$r^i = y - Dx^i. \tag{1.10}$$

Une alternative à OMP, nommée *Stagewise Orthogonal Matching Pursuit* (StOMP) [DTDS12], permet de sélectionner à chaque étape plusieurs atomes du dictionnaire, dont la corrélation avec le résidu courant est supérieure à un certain seuil, plutôt que de sélectionner uniquement l'atome le plus corrélé.

#### 1.1.2.3 *Basis Pursuit*

Afin de simplifier le problème précédent où la $norme - l_0$ est contraignante pour l'optimisation, une approche différente consiste à remplacer la $norme{-}l_0$ par la $norme-$

$l_1$. La $norme - l_1$ est définie par : $||x||_1 = \sum_{i=1}^{K} |x_i|$. Ce problème, connu sous le nom de *Basis Pursuit* (BP) [CDS98], s'écrit alors :

$$\min_x ||x||_1, \text{ sous la contrainte } y = Dx. \tag{1.11}$$

Cela permet d'obtenir un problème d'optimisation qui peut être résolu par des routines standard.

Ce problème peut également être formulé sous une forme Lagrangienne en cherchant à minimiser l'erreur de reconstruction du signal $y$ avec une contrainte sur la $norme - l_1$ de x :

$$\min_x(||y - Dx||_2^2 + \lambda||x||_1) \tag{1.12}$$

avec $\lambda$ un paramètre de régularisation.

#### 1.1.2.4 Autres méthodes de décomposition parcimonieuse

Il existe de nombreuses autres méthodes de décomposition parcimonieuse. On citera notamment la méthode OOMP (*Optimized Orthogonal Matching Pursuit*) [RNL02], évolution de la méthode OMP avec un critère de sélection de chaque atome différent, ou encore les algorithmes de type CMP (*Complementary Matching Pursuit*) [RG08], agissant de manière complémentaire aux algorithmes de type MP en sélectionnant les $(K-1)$ atomes du dictionnaire à exclure de l'approximation plutôt que l'atome à ajouter. Enfin, le lecteur peut également se reporter à [BD08], où des méthodes itératives de seuillage sont présentées.

### 1.1.3 La parcimonie structurée

Les algorithmes gloutons tels que MP ou OMP, bien que capables d'offrir de bonnes performances de reconstruction, sont relativement complexes du fait des comparaisons nécessaires à chaque itération avec chaque atome du dictionnaire. Ainsi, augmenter la taille du dictionnaire impacte directement la complexité de la décomposition. Des méthodes de décomposition utilisant la notion de parcimonie structurée ont donc vu le jour afin d'offrir un meilleur compromis entre qualité de représentation et complexité. Ces méthodes cherchent à structurer le dictionnaire, par exemple sous forme d'arbre, en regroupant certains atomes ensemble et utilisent alors un algorithme de décomposition parcimonieuse et structurée adapté à ce dictionnaire.

Les auteurs de [JVF06] proposent un algorithme de poursuite basé sur une structure en arbre d'un dictionnaire arbitraire afin d'offrir un bon compromis entre complexité et performances d'approximation. Pour cela, les atomes du dictionnaire jugés similaires sont regroupés, d'après une mesure de similarité basée sur la cohérence, sous forme de "molécules". D'abord appliqué aux atomes, le regroupement est ensuite appliqué aux molécules, de façon à obtenir une structure en arbre. L'arbre est ainsi construit de bas en haut : les atomes du dictionnaire sont groupés en molécules, puis les molécules sont ensuite elles-mêmes regroupées entre elles jusqu'à finalement n'obtenir plus qu'un nœud correspondant à la racine de l'arbre (Fig. 1.2). L'algorithme de décomposition

utilise ensuite cet arbre comme un arbre de décision. Comme le MP, l'algorithme cherche à chaque itération le meilleur atome pour approximer le résidu courant. Mais plutôt que de tester de manière exhaustive les atomes du dictionnaire, l'algorithme de décomposition utilise la structure en arbre, où les atomes similaires ont été regroupés. En partant du nœud racine de l'arbre, l'algorithme choisit à chaque nœud, le nœud fils le plus corrélé au signal à approximer, jusqu'à atteindre une feuille de l'arbre correspondant à un atome du dictionnaire, alors choisi dans la décomposition. Le fait de ne parcourir qu'une partie de l'arbre apporte ainsi un gain en complexité. Les auteurs montrent que ce gain en complexité n'entraîne en général pas une forte pénalité sur les performances d'approximation. Ils obtiennent par exemple une erreur de représentation comparable pour un temps d'exécution divisé par 5.

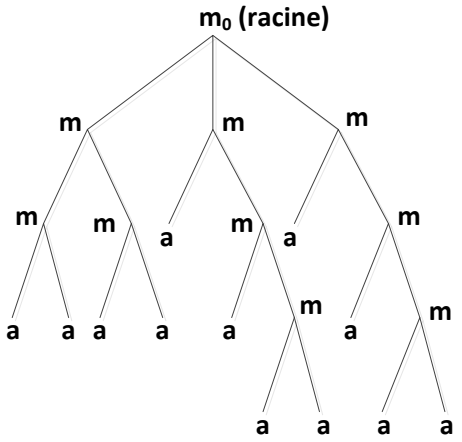



FIGURE 1.2 – Arbre construit pour un dictionnaire de 12 atomes. Les atomes (a) sont présents aux feuilles de l'arbre. Les nœuds de l'arbre correspondent à des molécules (m).

Afin de réduire encore davantage la complexité de la méthode précédente, ainsi que la mémoire nécessaire au stockage du dictionnaire, une alternative est présentée dans [WWHG09]. S'inspirant de [JVF06], les auteurs construisent un dictionnaire structuré en arbre mais remplacent les molécules par des atomes du dictionnaire afin de réduire l'empreinte mémoire du dictionnaire. Ainsi, à la place de trouver les atomes seulement sur les feuilles de l'arbre, chaque nœud de l'arbre est ici constitué par un atome du dictionnaire. De plus, pour réduire la complexité de l'algorithme, la similarité des atomes, permettant de les lier entre eux, est estimée par la distance entre des paramètres choisis pour caractériser les atomes. Des paramètres de translation, rotation et des facteurs multiplicatifs par rapport à deux atomes générateurs sont choisis pour caractériser chaque atome. Quant à l'algorithme de décomposition, il va sélectionner à chaque itération l'atome correspondant au produit scalaire maximal. Mais les comparaisons vont se limiter à une partie de l'arbre. Elles sont d'abord opérées avec les atomes au premier niveau, et l'atome optimal et le coefficient associé sont sauvegardés. Puis les comparaisons se font ensuite sur les atomes fils de l'atome optimal du niveau

précédent, jusqu'à atteindre les feuilles de l'arbre. En comparant alors les différents coefficients sauvegardés, l'atome associé au coefficient maximal est considéré comme le meilleur et choisi pour la décomposition. A noter que selon les performances souhaitées ou les contraintes de complexité, il est possible de stopper la recherche d'un atome avant d'atteindre les feuilles de l'arbre, c'est-à-dire à un niveau supérieur. Cet algorithme permet donc d'atteindre comme [JVF06] un bon compromis entre performances et efficacité d'implémentation. A titre indicatif, l'algorithme peut atteindre une vitesse d'exécution environ 57 fois plus rapide que celle de l'algorithme MP pour une baisse de PSNR moyenne de 0.6 dB.

L'idée de regrouper les atomes en molécules apparaît également dans [Dau06] dans un contexte de codage de signaux audio. La publication décrit une extension de l'algorithme MP nommée *Molecular Matching Pursuit* (MMP). Le dictionnaire utilisé est composé de la concaténation de deux sous-dictionnaires distincts : la base des atomes de la *Modified Discrete Cosine Transform* (MDCT) et la base des atomes de la *Discrete Wavelet Transform* (DWT). Les molécules correspondent alors au regroupement d'atomes provenant de l'un ou de l'autre sous-dictionnaire. L'algorithme MMP identifie à chaque itération la molécule la plus significative grâce à deux indices de corrélation correspondant aux deux domaines MDCT et DWT. La molécule complète (composée de plusieurs atomes) est alors retirée du résidu courant. Ainsi, la complexité est réduite, au prix d'une légère sous-optimalité du taux de convergence de l'erreur d'approximation.

Au contraire des méthodes précédentes cherchant à diminuer la complexité du MP, des techniques de recherche en arbre plus complexes sont appliquées dans [CR01] aux variantes du MP afin d'en améliorer la qualité de représentation. Elles permettent de tester davantage de solutions et donc de réduire la sous-optimalité de l'algorithme, mais au prix d'une augmentation de la complexité. Deux algorithmes de recherche des meilleurs atomes sont testés : MP :K [1] et MP :M-L. A chaque itération des algorithmes de poursuite (MP, OMP), seul l'atome le plus corrélé au résidu courant est sélectionné, mais l'information de corrélation entre le résidu courant et les autres atomes n'est pas utilisée. Les auteurs proposent avec le MP :K d'utiliser cette information afin d'évaluer d'autres solutions que celle offerte par la sélection d'un atome uniquement. Ils proposent donc de garder les K atomes les plus corrélés au résidu courant. K résidus sont alors calculés. Puis pour chacun d'entre eux, on cherchera de mêmes les K meilleurs atomes, pour ainsi calculer $K^2$ résidus au niveau suivant. L'arbre est ainsi construit jusqu'au niveau désiré. Chaque branche de l'arbre donne alors une solution de décomposition possible du signal $y$ et la meilleure est gardée. A noter que parmi toutes ces solutions se trouve celle du MP classique et que l'algorithme MP :K est équivalent au MP dans le cas où $K = 1$. MP :M-L constitue une alternative au MP :K afin de limiter le nombre de solutions testées. En effet, seuls les M nœuds offrant la plus petite norme des résidus sont retenus par niveaux, les autres sont rejetés. De cette façon, certaines branches sont

1. Le paramètre K utilisé dans ce paragraphe diffère du paramètre K utilisé tout au long de la thèse pour indiquer la taille d'un dictionnaire

élaguées et seulement M sont prolongées à chaque niveau, jusqu'à atteindre le niveau L. Moins complexe que le MP :K, cet algorithme est aussi plus sous-optimal du fait de l'élagage des branches à chaque niveau. L'algorithme MP :K avec K=2 permet par exemple d'améliorer les performances du MP classique tout en limitant l'augmentation de complexité par rapport à des valeurs de K plus grandes.

Une autre variante de l'OMP basée arbre est explorée dans [LD06] et cherche une représentation parcimonieuse en arbre des signaux dans le domaine ondelettes afin d'obtenir une meilleure reconstruction. Le but est ici de tirer profit de la structure en arbre des coefficients d'ondelettes les plus significatifs.

## 1.2 Apprentissage de dictionnaire

La qualité d'une représentation parcimonieuse d'un signal dépendant fortement de l'espace dans lequel il est représenté, l'apprentissage du dictionnaire est un point clé pour rendre les atomes le plus efficace possible pour un type de données particulier. Il a été montré qu'un dictionnaire appris a le pouvoir d'offrir une meilleure qualité de reconstruction qu'un dictionnaire prédéfini, par exemple dans le cadre de la compression d'images avec des gains pouvant atteindre 1 à 2 dB [AEB06]. Cette section aborde ainsi le problème de l'apprentissage de dictionnaire. Plusieurs algorithmes y sont présentés, apprenant des dictionnaires sans contrainte, des dictionnaires eux-mêmes parcimonieux, ou bien des dictionnaires avec une contrainte de non-négativité. Enfin, le cas des dictionnaires structurés est traité.

### 1.2.1 Formulation du problème

L'apprentissage de dictionnaire consiste à apprendre un dictionnaire $D \in \mathbb{R}^{n\times K}$ sur un ensemble de $N$ vecteurs d'apprentissage $Y \in \mathbb{R}^{n\times N}$, chacun de dimension $n$, sous certaines contraintes de parcimonie. Chacune des colonnes du dictionnaire $d_k, k = 1, ..., K$, appelée atome, est normalisée, c'est-à-dire de $norme - l_2$ unitaire : $||d_k||_2 = 1, \forall k = 1, ..., K$. L'objectif est donc de trouver le dictionnaire optimal permettant de représenter efficacement l'ensemble des vecteurs d'apprentissage, ou vecteurs d'entraînement, de façon parcimonieuse, de telle sorte que $Y \approx DX$, avec $X \in \mathbb{R}^{K\times N}$ une matrice dont chaque colonne $x_i, i = 1, ..., N$ est parcimonieuse.

Le problème est donc le suivant :

$$\min_{D,X} ||Y - DX||_F^2, \text{ sous les contraintes } ||x_i||_0 \leq L \ \forall i \text{ et } ||d_k||_2 = 1 \ \forall k \tag{1.13}$$

avec $L > 0$ la contrainte de parcimonie de chaque colonne de $X$, c'est-à-dire le nombre maximal de valeurs non nulles dans chaque colonne $x_i, i = 1, ..., N$, et $||.||_F$ la norme de Frobenius : $||A||_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|^2}$.

Afin de résoudre ce problème, les méthodes d'apprentissage de dictionnaire suivent souvent le même schéma composé de deux étapes majeures : une étape de représentation

parcimonieuse dite de "codage" parcimonieux et une étape de mise à jour du dictionnaire, qui sont itérées jusqu'à atteindre un critère d'arrêt.

L'étape dite de "codage" parcimonieux cherche à trouver la matrice de coefficients $X$ correspondant à la représentation parcimonieuse des vecteurs d'apprentissage $Y$ sur le dictionnaire $D$, fixe à cette étape. Le problème peut ainsi s'écrire :

$$\min_{x_i} ||y_i - Dx_i||_2^2, \text{ sous la contrainte } ||x_i||_0 \leq L \; \forall i \tag{1.14}$$

avec $L$ la contrainte de parcimonie sur chaque vecteur $x_i$.

Cela revient à un problème de décomposition parcimonieuse (problème 1.3) qui peut être résolu par les algorithmes de décomposition présentés au paragraphe 1.1.2 : MP, OMP, ou BP si la parcimonie est contrôlée par la $norme - l_1$.

La seconde étape correspond à la mise à jour du dictionnaire $D$. Ayant calculé la matrice de coefficients $X$ précédemment, cette étape a pour but de trouver le dictionnaire optimal permettant de minimiser l'erreur de représentation des vecteurs d'apprentissage. Le problème s'écrit :

$$\min_{D} ||Y - DX||_F^2, \text{ sous la contrainte } ||d_k||_2 = 1 \; \forall k. \tag{1.15}$$

Différentes méthodes de mise à jour du dictionnaire ont été proposées et nous allons en présenter quelques unes dans les paragraphes suivants.

Quant au critère d'arrêt, il peut par exemple être une erreur de représentation des vecteurs d'apprentissage à atteindre, un test sur la convergence de cette erreur, ou encore un nombre maximum d'itérations fixé.

### 1.2.2 Méthode des Directions Optimales (MOD)

La "Méthode de Directions Optimales" (*Method of Optimal Directions* (MOD)) [EAH99] est inspirée de l'"Algorithme de Lloyd Généralisé" (*Generalized Lloyd Algorithm* (GLA)) utilisé dans le cadre de la construction de tables de code pour la quantification vectorielle [GG92]. MOD est une méthode d'apprentissage de dictionnaire itérative qui tour à tour, approxime chaque vecteur d'entraînement grâce au dictionnaire courant avec une contrainte de parcimonie (problème 1.14), et met à jour le dictionnaire. L'approximation est typiquement réalisée avec un algorithme de poursuite tel que MP ou OMP. Pour la mise à jour du dictionnaire, les auteurs cherchent à réduire l'erreur quadratique entre les vecteurs d'apprentissage et leur représentation parcimonieuse sur le dictionnaire : $\min_{D} ||Y - DX||_F^2$.

Pour cela, le dictionnaire à l'itération $i$ est évalué ainsi :

$$D^i = D^{i-1} + \Delta \tag{1.16}$$

avec $\Delta = RX^T(XX^T)^{-1}$ (voir [ERKD99] et [EAH00] pour le détail des calculs), où $R$ représente la matrice de résidu et $X$ la matrice de coefficients à l'itération $i-1$, tel que $R = Y - D^{i-1}X$. $\Delta$ correspond aux directions optimales pour la mise à jour du dictionnaire dans le cadre du problème à résoudre, d'où le nom de la méthode.

On a donc :

$$\begin{aligned} D^i &= D^{i-1} + RX^T(XX^T)^{-1} \\ &= D^{i-1} + (Y - D^{i-1}X)X^T(XX^T)^{-1} \\ &= D^{i-1} + (YX^T - D^{i-1}XX^T)(XX^T)^{-1} \\ &= D^{i-1} + YX^T(XX^T)^{-1} - D^{i-1} \end{aligned} \tag{1.17}$$

soit :

$$D^i = YX^T(XX^T)^{-1}. \tag{1.18}$$

De cette façon, la mise à jour du dictionnaire est faite globalement sur l'ensemble du dictionnaire. Une normalisation des colonnes du dictionnaire est ensuite réalisée. Mais l'inversion matricielle que cette mise à jour requiert implique une complexité importante, en particulier pour de larges dictionnaires, car la matrice $XX^T$ est de grande taille (à savoir $K \times K$).

### 1.2.3 Algorithme K-SVD

L'algorithme K-SVD [AEB06] permet d'apprendre un dictionnaire sur-complet adapté à un ensemble de signaux d'apprentissage. Cet algorithme est une généralisation de l'algorithme K-means. En effet, l'algorithme K-means cherche la table de codes $C$ pour représenter les données $Y$ en résolvant :

$$\min_{C,X} ||Y - CX||_F^2, \text{ sous la contrainte } \forall i, x_i = e_k \textit{ pour un certain } k \tag{1.19}$$

avec $e_k$ un vecteur nul à l'exception d'une valeur 1 à la ligne $k$.

L'algorithme permet ainsi de représenter chaque vecteur de $Y$ par un seul vecteur de $C$ avec un coefficient associé unitaire. L'algorithme K-SVD généralise le problème de l'algorithme K-means (1.19) en traitant le problème (1.13), de telle sorte que chaque vecteur de $Y$ soit représenté par une combinaison linéaire d'atomes du dictionnaire $D$ avec des coefficients associés non unitaires.

Tout comme MOD, c'est une méthode qui itère entre l'étape de codage parcimonieux des vecteurs d'entraînements sur le dictionnaire courant, pouvant être réalisée par tout algorithme de poursuite tel que les algorithmes MP ou OMP, et l'étape de mise à jour du dictionnaire permettant d'améliorer la représentation des données. Cette mise à jour est réalisée atome par atome, en considérant les autres atomes fixes, grâce à l'algorithme de Décomposition en Valeurs Singulières (*Singular Value Decomposition* (SVD)) [KL80].

Ainsi, à chaque itération, chaque colonne $d_k$ du dictionnaire est mise à jour de la façon suivante dans le but de réduire l'erreur de reconstruction. D'abord, le groupe des vecteurs de $Y$ utilisant cet atome $d_k$ est défini par $\omega_k = \{i | 1 \leq i \leq N, x_T^k(i) \neq 0\}$, où $x_T^k$ correspond à la ligne n°$k$ de $X$. Puis une matrice d'erreur est calculée :

$$E_k = Y - \sum_{j \neq k} d_j x_T^j. \tag{1.20}$$

Cette matrice représente l'erreur de représentation des données $Y$ lorsque l'atome $d_k$ du dictionnaire est retiré. Cette matrice est ensuite restreinte à la matrice $E_k^R$ en ne sélectionnant que les colonnes de $E_k$ correspondant à $\omega_k$, c'est-à-dire en se limitant aux signaux utilisant l'atome $d_k$ dans leur décomposition. On cherche alors à mettre à jour $d_k$ de telle sorte qu'il représente au mieux $E_k^R$. Pour cela, la SVD est appliquée à $E_k^R$ : $E_k^R = U\Delta V^T$. La première colonne de $U$ est alors choisie pour être la mise à jour de l'atome $d_k$ et le vecteur de coefficients associé $x_R^k$ est également mis à jour comme la première colonne de $V$ multipliée par $\Delta(1,1)$.

L'algorithme K-SVD, efficace et moins complexe que MOD, a été très utilisé par la communauté dans diverses applications telles que la compression [BE08], le débruitage [EA06b], ou encore la classification [MBP+08a, JLD11]. Comme nous allons le voir par la suite, plusieurs variantes ont également été développées avec des contraintes de parcimonie sur le dictionnaire ou bien de non-négativité.

### 1.2.4 Dictionnaire parcimonieux

Une variante de l'algorithme K-SVD, nommée *Sparse K-SVD* [RZE10], cherche à apprendre un dictionnaire avec une contrainte de parcimonie sur le dictionnaire lui-même. Le modèle de dictionnaire parcimonieux suggère que chacun de ses atomes peut également être représenté de façon parcimonieuse sur un dictionnaire de base $\Phi$.

Le dictionnaire s'écrit donc :

$$D = \Phi A \tag{1.21}$$

avec $\Phi$ le dictionnaire de base prédéfini et $A$ la représentation parcimonieuse des atomes de $D$. $A$ possède donc un nombre fixé de valeurs non-nulles à ne pas dépasser. La représentation d'un tel dictionnaire est ainsi à la fois flexible, puisque $A$ est appris sur un ensemble de vecteurs d'apprentissage, compacte et efficace, la parcimonie contrainte de $A$ réduisant la complexité d'apprentissage. La propriété de parcimonie du dictionnaire peut également être intéressante dans le cas où l'on souhaiterait transmettre le dictionnaire.

Le problème devient alors :

$$\min_{A,X} ||Y - \Phi AX||_F^2, \text{ sous les contraintes } ||x_i||_0 \leq L \; \forall i, ||a_k||_0 \leq T \text{ et } ||\Phi a_k||_2 = 1 \forall k \tag{1.22}$$

avec $L$ la contrainte de parcimonie sur chaque vecteur $x_i$ et $T$ la contrainte de parcimonie sur chaque colonne $a_k$.

L'algorithme d'apprentissage d'un dictionnaire parcimonieux suit le même schéma que pour K-SVD : une étape de codage parcimonieux et une étape de mise à jour du dictionnaire $A$, réalisée colonne par colonne, itérées un nombre d'itérations fixé. Le changement notable avec K-SVD se situe dans l'étape de mise à jour de chaque colonne du dictionnaire. En effet, chaque atome de $D$ peut s'écrire sous la forme $d = \Phi a$ avec une contrainte de parcimonie $||a||_0 \leq T$. L'atome à mettre à jour $a$ et la ligne de coefficients correspondante $x^T$ sont donnés par la résolution du problème suivant :

$$\{a, x\} := \arg\min_{a,x} ||E - \Phi a x^T||_F^2, \text{ sous les contraintes } ||a||_0 \leq T \text{ et } ||\Phi a||_2 = 1 \tag{1.23}$$

avec $E$ défini comme $E_k^R$ dans l'algorithme du K-SVD, c'est-à-dire une matrice d'erreur sans l'atome $a$ et restreinte à $I$, les indices des signaux dans $Y$ utilisant l'atome $a$, et $x^T$ la ligne de coefficients dans $X$ correspondant à l'atome $a$ et restreinte à $I$.

Pour résoudre ce problème, l'approche utilisée est une optimisation alternée de $a$ et $x$. Après simplifications, l'optimisation selon $a$ s'écrit :

$$a := \arg\min_a ||Ex - \Phi a||_2^2, \text{ sous la contrainte } ||a||_0 \leq T \tag{1.24}$$

avec $x$ précédemment normalisé. Cela correspond alors à un problème de codage parcimonieux. Puis $a$ est normalisé par : $a := \frac{a}{||\Phi a||_2}$. Et l'optimisation selon $x$ s'écrit :

$$x := E^T \Phi a \tag{1.25}$$

### 1.2.5 Apprentissage avec contrainte de non-négativité

#### 1.2.5.1 Sparse Non-negative Matrix Factorization

La Non-negative Matrix Factorization (NMF) [LS00] représente une approximation linéaire et non-négative des données. Son but est d'approximer une matrice positive $Y$ par le produit de deux matrices positives : $Y \approx DX$. Le fait d'ajouter cette contrainte de non-négativité permet d'obtenir une représentation purement additive. Deux fonctions de coût à optimiser peuvent être utilisées, utilisant soit le carré de la norme de Frobenius, soit la divergence de Kullback–Leibler.

La première est la plus communément utilisée et le problème peut alors s'écrire :

$$\min_{D,X} ||Y - DX||_F^2, \text{ sous les contraintes } Y, D, X \geq 0. \tag{1.26}$$

Ce problème est résolu par l'utilisation de deux équations de mise à jour multiplicative, appliquées au dictionnaire $D$ et aux coefficients $X$ :

$$D_{hi} = D_{hi} \frac{(YX^T)_{hi}}{(DXX^T)_{hi} + \epsilon}, \quad X_{ij} = X_{ij} \frac{(D^T Y)_{ij}}{(D^T DX)_{ij} + \epsilon} \tag{1.27}$$

avec $\epsilon$ une faible constante permettant de ne pas diviser par 0 et $D_{hi}$ (ou $X_{ij}$) représentant l'élément à la ligne $h$ (ou $i$) et à la colonne $i$ (ou $j$) de la matrice $D$ (ou $X$).

L'algorithme NMF avec contraintes de parcimonie (ou Sparse NMF) [Hoy04] permet d'ajouter à l'algorithme NMF des contraintes de parcimonie à la fois sur la matrice de coefficients $X$ et sur le dictionnaire $D$.

La mesure de parcimonie définie dans cet article est basée sur une relation entre les normes $l_1$ et $l_2$ :

$$parcimonie(x) = \frac{\sqrt{n} - \frac{\sum_i |x_i|}{\sqrt{\sum_i x_i^2}}}{\sqrt{n} - 1} \tag{1.28}$$

où $n$ représente la dimension de $x$. Ainsi, cette mesure de parcimonie est égale à 1 si et seulement si $x$ contient une seule valeur non nulle, et 0 si et seulement si toutes les

composantes de $x$ sont égales. Cette mesure quantifie à quel point l'énergie d'un vecteur est concentrée sur seulement quelques composantes.

Après une initialisation des matrices $D$ et $X$, ces matrices sont alternativement mises à jour pendant un certain nombre d'itérations. Si aucune contrainte de parcimonie n'est appliquée à la matrice à mettre à jour ($D$ ou $X$), alors la mise à jour est effectuée comme pour la NMF (équations 1.27). Dans le cas contraire, la matrice est mise à jour de façon particulière en imposant une contrainte de non-négativité et en jouant sur la $norme - l_1$ de façon à satisfaire la contrainte de parcimonie.

Mais l'algorithme applique la contrainte de parcimonie sur les lignes de $X$ et non sur les colonnes comme nous le voudrions, ce qui implique que la contrainte de parcimonie n'est pas appliquée sur la représentation de chaque vecteur $y$ dans $Y$.

#### 1.2.5.2 Non-Negative K-SVD

Le Non-Negative K-SVD (NN-K-SVD) [AEB05] est une variante de l'algorithme K-SVD apprenant un dictionnaire avec une contrainte de non-négativité sur ses atomes et sur les coefficients correspondants. L'algorithme NN-K-SVD peut donc être utilisé dans le cadre des problèmes de NMF [LS00] où le signal est représenté par un modèle additif étant donné que les atomes et les coefficients sont positifs. En d'autres termes, la combinaison linéaire des atomes du dictionnaire est une somme de termes uniquement positifs.

Afin d'adapter l'algorithme K-SVD à l'apprentissage d'un dictionnaire positif et le calcul d'une matrice de coefficients positive, des changements sont appliqués lors des étapes de codage parcimonieux et de mise à jour du dictionnaire.

A l'étape de codage parcimonieux, un algorithme de poursuite permettant d'obtenir une décomposition positive doit être utilisé. Le problème devient pour $i = 1, ..., N$ :

$$\min_x ||y_i - Dx||_2^2, \text{ sous les contraintes } ||x||_0 \leq L \text{ et } \forall j \; x_j \geq 0. \tag{1.29}$$

Une méthode itérative présentée dans [Hoy02], résolvant une variation du problème de BP pour des décompositions non-négatives, est utilisée :

$$x^{t+1} = x^t .* (D^T y)./(D^T D x^t + \lambda) \tag{1.30}$$

où .* et ./ représentent des opérations terme à terme sur les différents vecteurs.

Une modification est cependant apportée à cette méthode afin de trouver une décomposition avec un nombre fixé de coefficients non nuls $L$. Pour cela, après plusieurs itérations, les indices des $L$ plus grands coefficients sont sélectionnés, et le vecteur de données est approximé seulement par les $L$ atomes correspondants en résolvant un problème de moindres carrés avec une contrainte de non-négativité sur les coefficients :

$$\min_x ||y - D^L x|| \text{ sous la contrainte } x \geq 0 \tag{1.31}$$

où $D^L$ représente une sous-matrice du dictionnaire $D$ incluant seulement les $L$ atomes sélectionnés.

Au niveau de la mise à jour du dictionnaire, chaque atome mis à jour doit être forcé à rester positif. Comme pour K-SVD, la matrice d'erreur $E_k^R$ est calculée. Afin de mettre à jour chaque atome $d_k$ ainsi que la ligne de coefficients correspondante $x^k$, le problème devient :

$$\min_{d_k, x^k} ||E_k^R - d_k x^k||, \text{ sous les contraintes } d_k, x^k \geq 0. \tag{1.32}$$

Un algorithme itératif est utilisé pour résoudre ce problème. La solution issue d'une SVD est choisie comme solution initiale, en mettant à 0 les valeurs négatives. Puis $d_k$ et $x^k$ sont alternativement mis à jour en forçant à 0 toute valeur négative :

$$d_k = \frac{E_k^R x^k}{x^{k\prime} x^k} \;\; et \;\; x^k = \frac{d_k{}' E_k^R}{d_k{}' d_k}. \tag{1.33}$$

Après les différentes itérations, $d_k$ est normalisé en le divisant par un scalaire et $x^k$ est multiplié par le même scalaire.

### 1.2.6 Apprentissage de dictionnaire en ligne

Une méthode d'apprentissage de dictionnaire en ligne, basée sur des approximations stochastiques et adaptée à des applications de codage parcimonieux, a été proposée par Mairal et al. [MBPS10]. Cette méthode considère que l'ensemble des vecteurs d'apprentissage est composé d'échantillons indépendants et identiquement distribués d'une distribution $p(y)$, avec $y$ une variable aléatoire telle que $y \in \mathbb{R}^n \sim p(y)$. Ainsi, l'algorithme traite à chaque itération $t$ un élément $y_t$ de la distribution $p(y)$ et alterne entre une étape de codage parcimonieux permettant de calculer la représentation $x_t$ de $y_t$ sur le dictionnaire de l'itération précédente $D_{t-1}$, et une étape de mise à jour du dictionnaire calculant le nouveau dictionnaire $D_t$.

Lors de l'étape de codage parcimonieux, les coefficients $x_t$ correspondant à la représentation de $y_t$ sont calculés, en utilisant l'algorithme LARS [EHJ$^+$04], en résolvant :

$$x_t = \arg\min_x \frac{1}{2}||y_t - D_{t-1}x||_2^2 + \lambda||x||_1. \tag{1.34}$$

Les matrices $A = [a_1, ..., a_K] \in \mathbb{R}^{K \times K}$ et $B = [b_1, ..., b_K] \in \mathbb{R}^{n \times K}$, contenant les informations des coefficients passés, c'est-à-dire obtenus aux itérations précédentes, sont mises à jour :

$$A_t = A_{t-1} + x_t x_t^T \tag{1.35}$$

$$B_t = B_{t-1} + y_t x_t^T \tag{1.36}$$

avec $A_0 = 0$ et $B_0 = 0$.

Puis le dictionnaire est mis à jour à son tour de telle sorte que :

$$\begin{aligned} D_t &= \arg\min_{D \in C} \frac{1}{t} \sum_{i=1}^{t} (\frac{1}{2}||y_i - Dx_i||_2^2 + \lambda||x_i||_1) \\ &= \arg\min_{D \in C} \frac{1}{t} (\frac{1}{2} Tr(D^T D A_t) - Tr(D^T B_t)) \end{aligned} \tag{1.37}$$

avec $C = \{D \in \mathbb{R}^{n \times K},$ sous la contrainte $\forall k = 1, ..., K, d_k^T d_k \leq 1\}$.

Ce problème est résolu par un algorithme de descente ("block-coordinate descent") (voir [Ber99]), le dictionnaire calculé à l'itération précédente $D_{t-1}$ servant d'initialisation, en répétant la mise à jour de chaque atome $d_k$ du dictionnaire individuellement de la façon suivante, jusqu'à convergence :

$$u_k = \frac{1}{A(k,k)}(b_k - Da_k) + d_k \tag{1.38}$$

$$d_k = \frac{1}{\max(||u_k||_2, 1)} u_k. \tag{1.39}$$

Il a été montré que cet algorithme est plus rapide que les alternatives traitant l'ensemble des vecteurs d'apprentissage simultanément, telles que MOD ou K-SVD, sur des ensembles de données importants pouvant atteindre plusieurs millions de vecteurs d'entraînement. De plus, l'aspect en ligne apporte une adaptabilité à l'algorithme qui peut s'adapter à tout nouveau signal d'apprentissage. Enfin, l'algorithme peut être adapté pour traiter par exemple des problèmes de NMF.

### 1.2.7 *Image Signature Dictionary* (ISD)

L'idée des ISD [AE08] est de représenter une image par une image de plus petite taille. Cette idée provient du concept d'épitomes, où *épitome* est le nom donné à cette représentation de petite taille apprise sur une image d'entrée. Un épitome est donc une représentation plus compacte d'une image contenant les caractéristiques de texture de cette image. Une image peut ainsi être représentée par son épitome associé à une carte de correspondances entre des patchs de l'épitome et des patchs de l'image.

Une approche utilisant un modèle probabiliste a d'abord été explorée [JFK03]. L'épitome est alors décrit par un épitome de moyennes et un épitome de variances et l'image reconstruite est générée d'après la carte de correspondance en copiant les pixels appropriés de l'épitome de moyenne et en ajoutant du bruit gaussien dont la valeur est donnée par l'épitome de variance. L'épitome est appris en itérant une étape de mise à jour des correspondances entre les pixels de l'épitome et ceux de l'image, et une étape de mise à jour des épitomes de moyenne et de variance. Ce modèle a par la suite été étendu au cas de la vidéo [CFJ08].

Une deuxième approche construit l'épitome comme un assemblage de patchs de texture de l'image. L'approche développée dans [CGT+11], et inspirée de [WWOH08], cherche à réduire la redondance présente dans l'image en cherchant des similarités entre les patchs de l'image. La construction de l'épitome est réalisée en trois étapes majeures. La première étape consiste à trouver des similitudes entre les patchs de l'image d'entrée et des blocs de cette même image, en utilisant un algorithme de "block matching" avec une distance euclidienne moyenne. La deuxième étape consiste à créer l'épitome, morceau par morceau en sélectionnant des patchs de texture de l'image. Le patch le plus représentatif est choisi en premier, puis l'épitome est progressivement étendu jusqu'à être capable de représenter tous les blocs de l'image. La troisième étape consiste à mettre à jour la carte de correspondances en cherchant, pour chaque bloc de l'image, le

meilleur patch correspondant au sein de l'épitome. Avec l'épitome et la carte de correspondances, il est ainsi possible de reconstruire une approximation de l'image originale.

Les ISD [AE08] peuvent être vus comme une généralisation des épitomes. En effet, là où la méthode des épitomes copie des patchs de l'épitome pour reconstruire l'image, la méthode des ISD représente chaque bloc de l'image par une combinaison linéaire de quelques patchs de l'ISD.

Définissons quelques notations propres au concept des ISD. Soit $D_S \in \mathbb{R}^{\sqrt{K}\times\sqrt{K}}$ un ISD. $d_S \in \mathbb{R}^K$ représente l'ISD mis sous forme d'une seule colonne. $y \in \mathbb{R}^n$ est un vecteur correspondant à un bloc de l'image, à représenter avec l'ISD. $C_{[k,l]} \in \mathbb{R}^{n\times K}$ est défini comme un opérateur linéaire permettant d'extraire un patch de taille $\sqrt{n}\times\sqrt{n}$ de l'ISD $D_S$ à la position $[k,l]$ (coin en haut à gauche du patch), c'est-à-dire que $C_{[k,l]}d_S$ est le patch extrait mis sous forme d'une colonne, que l'on peut donc assimiler à un atome. A noter que l'opération d'extraction de patch est considérée cyclique, c'est-à-dire que l'ISD est considéré périodique. Via ces notations, $y$ peut donc s'écrire comme une combinaison linéaire de patchs (atomes) de l'ISD $D_S$ de cette façon :

$$y = \sum_{k=1}^{\sqrt{K}}\sum_{l=1}^{\sqrt{K}} x_{[k,l]}C_{[k,l]}d_S. \tag{1.40}$$

En considérant touts les patchs de l'ISD se chevauchant mis sous la forme de colonnes, on obtient un dictionnaire $D \in \mathbb{R}^{n\times K}$ de $K$ atomes et on retrouve la formulation $y = Dx$. Un ISD représente ainsi de façon compacte un dictionnaire, étant donné que chaque patch de l'ISD correspond à un atome et que les patchs se chevauchent. L'approche est de plus moins complexe que les approches classiques.

Le problème revient alors à chercher à minimiser l'erreur de représentation des vecteurs d'apprentissage sur l'ISD avec une contrainte de parcimonie sur le nombre d'atomes utilisés par vecteur, soit le problème classique de l'apprentissage de dictionnaire sur $N$ vecteurs d'entraînement :

$$\hat{d_S} = \arg\min_{d_S} \sum_{i=1}^{N} E_i^2(d_S) \tag{1.41}$$

$$E_i^2(d_S) = \min_x ||y_i - \sum_{k=1}^{\sqrt{K}}\sum_{l=1}^{\sqrt{K}} x_{[k,l]}C_{[k,l]}d_S||_2^2 \text{ sous la contrainte } ||x||_0 \leq L,\ 1 \leq i \leq N \tag{1.42}$$

avec $x$ la concaténation des $K$ coefficients $x_{[k,l]}$.

L'approche utilisée pour résoudre ce problème consiste à classiquement itérer une étape de codage parcimonieux de l'ensemble des vecteurs d'apprentissage et une étape de mise à jour de l'ISD. Tout algorithme de poursuite peut être utilisé pour l'étape de codage parcimonieux, comme l'algorithme OMP ici choisi. Quant à la mise à jour du dictionnaire, elle peut être réalisée directement en une étape de façon similaire à la

méthode utilisée par l'algorithme MOD, en cherchant à annuler le gradient de l'erreur de représentation des vecteurs d'apprentissage sur l'ISD.

Une approche alternative de gradient stochastique propose de mettre à jour l'ISD après le codage parcimonieux de chaque vecteur d'entraînement $y_i$, de façon à accélérer la convergence de l'algorithme.

Enfin, un autre intérêt des ISD est de pouvoir apprendre des ISD multi-échelles offrant la possibilité de traiter des patchs d'entrainement de taille variable en les représentant par des patchs de l'ISD de la taille correspondante et dont le nombre peut varier également.

### 1.2.8 Dictionnaires structurés

#### 1.2.8.1 Structures en arbre

Afin de créer des relations entre les atomes, des dictionnaires structurés, notamment en arbre, ont vu le jour. Ces approches consistent à organiser les atomes du dictionnaire appris en une structure hiérarchique.

Dans [MV04], le dictionnaire est appris en cherchant à minimiser une fonction de coût composée de trois termes : un terme représentant le carré de l'erreur entre l'image originale et l'image reconstruite, un deuxième terme encourageant à trouver une représentation parcimonieuse et un troisième terme dont le but est de réduire l'introduction d'atomes n'ayant pas les caractéristiques voulues pour l'application considérée. Les atomes du dictionnaire appris sont regroupés en clusters organisés sous la forme d'une structure arborescente. L'arbre est construit par des utilisations récursives de l'algorithme K-means afin de diviser un groupe d'atomes en $M$ clusters. Chaque nœud de l'arbre possède ainsi $M$ fils et est associé à un cluster dont le centroïde représente les atomes présents dans le sous-arbre du nœud, les atomes étant situés aux feuilles de l'arbre. Un algorithme de poursuite pour structure arborescente est ensuite utilisé afin de sélectionner les atomes pour la représentation, en cherchant à chaque étape le meilleur chemin dans l'arbre jusqu'aux feuilles où se trouvent les atomes.

Le dictionnaire proposé dans [NNI09] est invariant par translation et contient des versions translatées d'éléments structurants, représentant les structures locales de l'image. Une structure en arbre binaire est utilisée pour apprendre les éléments structurants d'une image et déterminer combien en apprendre. En commençant par 1, le nombre d'éléments structurants à apprendre est doublé à chaque itération jusqu'à ce qu'un critère sur le taux de réduction de la fonction de coût soit satisfait. Les éléments structurants sont appris en alternant entre la mise à jour des coefficients pour chaque translation et la mise à jour des éléments structurels.

Les auteurs proposent dans [JMOB10b, JMOB10a] d'apprendre un dictionnaire dont les atomes sont hiérarchisés sous la forme d'un arbre, chaque nœud correspondant à un atome (Fig. 1.3). La structuration du dictionnaire est rendue possible par l'utilisation d'une norme hiérarchique favorisant la parcimonie, d'abord introduite par [ZRY09], en remplacement des classiques normes $l_0$ ou $l_1$. Cette norme favorise les dépendances entre atomes de telle sorte que les atomes choisis dans la décomposition

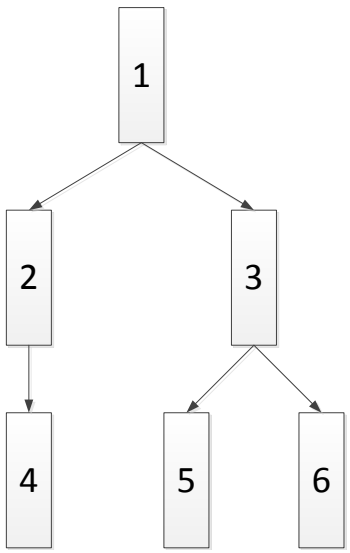



FIGURE 1.3 – Dictionnaire de 6 atomes structuré en arbre.

soient sélectionnés dans une même branche de l'arbre. Ainsi, un atome peut intervenir dans la décomposition seulement si ses ancêtres sont également dans la décomposition. De façon équivalente, si un atome ne fait pas partie de la décomposition, ses descendants non plus. L'algorithme d'apprentissage itère entre la mise à jour des coefficients via des méthodes proximales (en prenant en compte cette norme hiérarchique) et la mise à jour du dictionnaire, réalisée en suivant la procédure décrite dans [MBPS10].

Pour ces méthodes, la structure arborescente représente un unique dictionnaire, dont les atomes sont soit présents aux feuilles de l'arbre, soit aux nœuds. Nous allons maintenant nous intéresser à des méthodes apprenant des structures de dictionnaires où chaque nœud représente un dictionnaire.

#### 1.2.8.2 *Iteration-Tuned Dictionaries* (ITD)

Un dictionnaire structuré, mieux adapté aux algorithmes de décomposition parcimonieuse, a été introduit. Appelée *Iteration-Tuned Dictionary* (ITD), la structure proposée dans [ZGK$^+$10, Zep10] est composée d'un ensemble de dictionnaires répartis sur différents niveaux, chaque dictionnaire étant associé à une itération de l'algorithme de décomposition. Ainsi, à chaque itération, le choix de l'atome est effectué dans un nouveau dictionnaire, dont les atomes sont différents des atomes des dictionnaires utilisés aux itérations précédentes.

L'apprentissage de la structure est réalisé de haut en bas : chaque dictionnaire est appris sur des résidus du niveau précédent, avec un algorithme similaire à K-SVD [AEB06] avec une contrainte de parcimonie fixée à 1 atome. Les résidus utilisés pour apprendre le premier niveau correspondent donc aux données sources. Chaque dictionnaire $D$ est appris sur un ensemble de résidus $R$, constituant les vecteurs d'apprentissage de $D$, de

la façon suivante. Le dictionnaire $D$ de $K$ atomes est d'abord initialisé, par exemple par une sélection aléatoire de $K$ vecteurs d'apprentissage dans $R$. Puis, comme pour K-SVD, des itérations d'une étape de codage parcimonieux et d'une étape de mise à jour du dictionnaire atome par atome sont réalisées. Lors de l'étape de codage parcimonieux, réalisée par exemple avec l'algorithme MP pour une parcimonie de 1 atome, on recherche pour chaque vecteur de $R$ l'atome de $D$ qui lui est le plus corrélé (voir 1.4). Ainsi, chaque vecteur de $R$ est assigné à un seul atome $d_k, k = 1, ..., K$, de $D$. On peut alors définir pour chaque atome $d_k$ une matrice $R_k$ correspondant aux vecteurs de $R$ projetés sur $d_k$. Puis, lors de l'étape de mise à jour du dictionnaire, une SVD [KL80] est appliquée à chaque matrice $R_k$, et chaque atome $d_k$ correspondant prend la valeur du premier vecteur singulier de gauche de $R_k$, associé à la plus forte valeur singulière.

Un algorithme de décomposition adapté à cette structure sélectionne 1 atome par niveau, avec un algorithme tel que MP appliqué au dictionnaire de niveau courant pour une parcimonie de 1 atome. Le dictionnaire utilisé change donc à chaque itération de l'algorithme de décomposition. On descend ainsi dans la structure jusqu'à atteindre le nombre d'atomes souhaité pour la décomposition, ou jusqu'à ce qu'une certaine erreur d'approximation soit atteinte.

Une structure simple utilisant le concept d'ITD correspond à un dictionnaire structuré en niveaux, chaque niveau $i$ contenant un unique dictionnaire $D_i$, utilisé pour l'itération $i$ de l'algorithme de poursuite, et contenant des atomes différents de ceux des dictionnaires des niveaux précédents. Cette structure, nommée *Basic Iteration-Tuned Dictionary* (BITD) [ZGK$^+$10, Zep10], peut être vue comme une unique branche d'un arbre (Fig. 1.4 (a)).

A l'apprentissage, chaque dictionnaire est appris sur l'ensemble des résidus du niveau supérieur. A chaque niveau $i$, le dictionnaire $D_i$ est appris sur les résidus $R_{i-1}$ (en utilisant l'algorithme décrit au paragraphe précédent), avec $R_0 = Y$ correspondant aux vecteurs d'apprentissage initiaux. Puis chaque vecteur de $R_{i-1}$ est approximé par 1 atome de $D_i$ (avec l'algorithme MP) afin de calculer les résidus au niveau suivant $R_i = R_{i-1} - D_i X$, avec $X$ la matrice de coefficients dont chaque colonne contient une seule valeur non nulle. Les résidus $R_i$ sont ensuite utilisés pour apprendre le dictionnaire $D_{i+1}$ au niveau $i + 1$.

Pour la décomposition, 1 atome est sélectionné par niveau, dans l'unique dictionnaire appris à chaque niveau, en appliquant l'algorithme de MP pour une parcimonie de 1 atome à chaque niveau.

Une structure en arbre a également été créée sur le principe des ITD : le Tree-Structured Iteration-Tuned Dictionary (TSITD) [ZGK11, Zep10]. Cette structure comporte plusieurs dictionnaires par niveau, chaque dictionnaire à un niveau étant le fils d'un atome au niveau précédent (Fig. 1.4 (b)). Elle est donc composée de 1 dictionnaire de $K$ atomes au premier niveau, $K$ dictionnaires au second niveau, $K^2$ au troisième, etc.

L'apprentissage de la structure est effectué de haut en bas, chaque dictionnaire à un niveau étant appris cette fois sur un sous-ensemble de résidus du niveau supérieur.

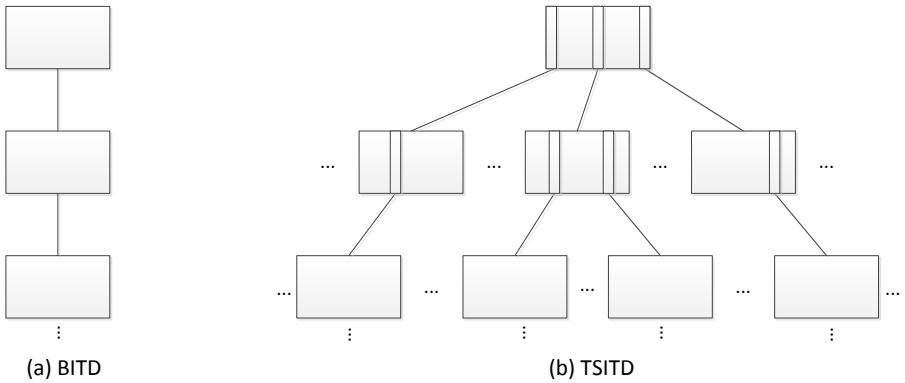



FIGURE 1.4 – Structures BITD (a) et TSITD (b).

Le dictionnaire au premier niveau $D_1$ est directement appris sur les données d'apprentissage $Y = R_0$. Puis ces données sont approchées par un vecteur uniquement de $D_1$ (avec un coefficient associé) pour calculer les résidus au niveau suivant $R_1$. Ces résidus sont alors partitionnés en $K$ ensembles selon l'atome utilisé pour l'approximation au premier niveau : $R_{1,k}, k = 1, ..., K$. Chaque ensemble de résidus $R_{1,k}$ permet ensuite d'apprendre un dictionnaire au deuxième niveau, etc.

L'algorithme de décomposition sélectionne toujours 1 atome par niveau et c'est l'indice de l'atome sélectionné au niveau $i$ qui indique dans quel dictionnaire chercher l'atome suivant au niveau $i + 1$, chaque atome à un niveau ayant un dictionnaire fils au niveau suivant. L'ensemble des atomes sélectionnés pour l'approximation d'un vecteur test est donc sélectionné le long d'une branche de l'arbre, en partant de la racine et en descendant à chaque itération de dictionnaire fils en dictionnaire fils. Ainsi, en décomposant un vecteur $y$ , si on regroupe les atomes sélectionnés jusqu'au niveau $i$ pour former une matrice des atomes sélectionnés $S = [d_1...d_i]$, ainsi que les coefficients correspondants dans le vecteur de coefficients $x = [x_1...x_i]^T$, alors on obtient l'approximation de $y$ après $i$ niveaux :

$$y \approx Sx \tag{1.43}$$

La structure *Iteration-Tuned and Aligned Dictionary* (ITAD) [ZGK11, Zep10] cherche à réduire l'empreinte mémoire de la structure arborescente TSITD pouvant contenir de nombreux dictionnaires. Ainsi, à chaque niveau $i$, il n'y a plus qu'un seul dictionnaire $D_i$. De plus, chaque atome $d_{i,k}$ ($i$ indiquant le niveau du dictionnaire et $k$ la place de l'atome au sein du dictionnaire de $K$ atomes) de $D_i$ est associé à une matrice d'alignement $\phi_{i,k} \in \mathbb{R}^{(n-i+1)\times(n-i)}$. Ces matrices d'alignement permettent une réduction de la dimension des résidus et des atomes à chaque niveau. Ainsi, la dimension des atomes de l'ITAD est de $n$ au premier niveau et décroit de 1 à chaque niveau. Elle est donc égale à $n-i+1$ au niveau $i$. Ces matrices d'alignement permettent également de mieux structurer les résidus à chaque niveau pour une meilleure représentation avec l'unique dictionnaire de chaque niveau.

L'apprentissage est encore une fois effectué de haut en bas, mais contrairement à TSITD, les résidus ne sont pas partitionnés à chaque niveau et chaque dictionnaire $D_i$ est appris sur l'ensemble des résidus du niveau précédent $R_{i-1}$. En plus des dictionnaires à apprendre, les matrices d'alignement doivent également être apprises. Ainsi, lors de l'étape de mise à jour de chaque atome d'un dictionnaire $D_i$, l'atome $d_{i,k}$ prend comme précédemment la valeur du premier vecteur singulier de gauche de $R_{i,k}$, et la matrice d'alignement $\phi_{i,k}$ est mise à jour avec tous les autres vecteurs singuliers de gauche, excepté le premier. Chaque vecteur de résidus mis à jour après le niveau $i$ par l'apport de l'atome $d_{i,k}$ est de plus multiplié par la transposée de la matrice d'alignement $\phi_{i,k}$ correspondant à $d_{i,k}$ (voir 1.44), de façon à obtenir $R_i$.

Pour décomposer un signal $y$ sur la structure ITAD, la mise à jour du résidu est effectuée ainsi à chaque niveau :

$$r^i = (\phi_{i,k})^T (r^{i-1} - x_i \, d_{i,k}) \tag{1.44}$$

avec $r^0 = y$. $d_{i,k}$ est l'atome sélectionné et $x_i$ le coefficient correspondant, calculés par les règles du MP (voir 1.4 et 1.5).

Afin de reconstruire le signal, on effectue l'opération suivante à chaque niveau :

$$\tilde{r}^{i-1} = x_i \, d_{i,k} + \phi_{i,k} \, \tilde{r}^i \tag{1.45}$$

avec par convention $\tilde{r}^L = 0$ et $L$ la parcimonie du bloc à reconstruire, $\tilde{r}^0$ donnant la reconstruction finale.

## 1.3 Applications

Les représentations parcimonieuses et l'apprentissage de dictionnaire sont aujourd'hui utilisés dans de nombreux domaines. Dans cette section nous insisterons sur trois domaines en particulier : le débruitage, la compression et la classification. Le débruitage est un domaine d'application fréquent des représentations parcimonieuses. Quant à la compression et la classification, ces deux domaines seront par la suite traités dans le cadre de cette thèse.

### 1.3.1 Débruitage

L'utilisation de représentations redondantes et de la parcimonie dans le cadre du débruitage s'est grandement développée cette dernière décennie, si bien que le problème de débruitage peut être traité comme un problème de représentations parcimonieuses sur un dictionnaire redondant. Différentes transformées ont alors été utilisées telles que les ondelettes [PSWS03], les *curvelets* [SCD02] ou encore les *contourlets* [ER06, MEZ05].

Plutôt que d'utiliser un ensemble de fonctions de bases prédéfinies (telles les *curvelets* ou les contourlets), les auteurs dans [EA06b, EA06a] proposent une approche basée sur les représentations redondantes et parcimonieuses sur des dictionnaires appris de

patchs se chevauchant, suivies par un algorithme de moyennage. Un dictionnaire est appris par l'algorithme K-SVD, soit sur des patchs d'images d'apprentissage de bonne qualité, soit sur des patchs de l'image bruitée elle-même.

Dans le premier cas, après avoir appris un dictionnaire avec K-SVD sur une collection de patchs d'image d'apprentissage, le problème de débruitage peut s'écrire :

$$\{\hat{x}_{ij}, \hat{Z}\} = \underset{x_{ij},Z}{\arg\min}\ \lambda||Z-Y||_2^2 + \sum_{ij}\mu_{ij}||x_{ij}||_0 + \sum_{ij}||Dx_{ij} - R_{ij}Z||_2^2 \tag{1.46}$$

Le premier terme de cette expression impose une proximité entre l'image bruitée $Y$ (de taille $\sqrt{N}\times\sqrt{N}$) et sa version débruitée à déterminer $Z$. Le deuxième et le troisième terme font en sorte que chaque patch $z_{ij} = R_{ij}Z$ de l'image $Z$, de taille $\sqrt{n}\times\sqrt{n}$, ait une représentation parcimonieuse sur le dictionnaire $D$ avec une erreur limitée. La matrice $R_{ij}$ est une matrice $n\times N$ permettant d'extraire le bloc $(ij)$ de l'image.

Après apprentissage du dictionnaire $D$, ce problème admet deux inconnues : les coefficients parcimonieux $x_{ij}$ de chaque patch $z_{ij}$ et l'image débruitée $Z$. Le problème est donc séparé en deux sous-problèmes. Après l'initialisation $Z = Y$, la représentation $x_{ij}$ optimale pour chaque patch est recherchée :

$$\hat{x}_{ij} = \underset{x}{\arg\min}\ \mu_{ij}||x||_0 + ||Dx - z_{ij}||_2^2 \tag{1.47}$$

Ce problème peut être résolu par l'algorithme OMP.

Une fois les représentations $x_{ij}$ fixées, $Z$ doit être mise à jour en résolvant :

$$\hat{Z} = \underset{Z}{\arg\min}\ \lambda||Z-Y||_2^2 + \sum_{ij}||D\hat{x}_{ij} - R_{ij}Z||_2^2 \tag{1.48}$$

dont la solution est de la forme :

$$\hat{Z} = (\lambda I + \sum_{ij}R_{ij}^T R_{ij})^{-1}(\lambda Y + \sum_{ij}R_{ij}^T D\hat{x}_{ij}) \tag{1.49}$$

Dans le second cas, l'apprentissage du dictionnaire est effectué directement sur l'image bruitée. L'algorithme d'apprentissage du dictionnaire est alors imbriqué dans l'algorithme de débruitage. Le problème du débruitage (1.46) est repris en ajoutant le dictionnaire $D$ dans les inconnues à déterminer. Les coefficients et le dictionnaire sont alors mis à jour alternativement et de façon itérative en utilisant les principes de l'algorithme K-SVD. Puis l'image débruitée est calculée en utilisant (1.49).

Le dictionnaire appris sur l'image bruitée elle-même donne globalement de meilleurs résultats que le dictionnaire appris sur la collection de patchs d'images d'apprentissage et que le dictionnaire DCT sur-complet. En comparaison de [PSWS03], la méthode présente un avantage en dessous d'un certain niveau de bruit, puis se détériore plus rapidement.

Ce travail a été étendu au cas du débruitage de vidéos dans [PE09].

Un modèle unifiant l'apprentissage de dictionnaire et le clustering structurel est proposé dans [DLZS11]. Basé sur l'observation que les coefficients parcimonieux non nuls ne sont pas distribués de façon aléatoire, un nouveau terme de régularisation basé clustering est ajouté à la fonction de coût (les autres termes étant les termes classiques du problème d'apprentissage de dictionnaire). Ce terme incite la représentation parcimonieuse de chaque patch à être proche du barycentre (centroïde) du cluster auquel elle appartient. Une plus grande parcimonie est ainsi attendue en exploitant la redondance structurelle des coefficients et donc les similitudes non locales. Cette méthode permet d'obtenir de meilleurs résultats que la méthode K-SVD [EA06b] et des résultats compétitifs par rapport à la méthode BM3D [DFKE07].

### 1.3.2 Compression

La compression est un autre cas d'application peu exploré jusqu'à présent des représentations parcimonieuses et de l'apprentissage de dictionnaire. Les méthodes de compression usuelles utilisent davantage des transformées prédéfinies. HEVC [SOHW12] utilise par exemple une transformée proche d'une DCT, déjà utilisée par JPEG [Wal91]. JPEG 2000 [TM02] utilise quand à lui une transformée en ondelettes. Nous allons désormais explorer quelques méthodes cherchant à apprendre une transformée spécifique. En effet, apprendre un dictionnaire pour une certaine classe d'images permet d'améliorer la parcimonie et donc la compressibilité de ces signaux.

Nous allons commencer par décrire quelques méthodes de compression nécessitant l'apprentissage de plusieurs dictionnaires, puis nous nous attarderons sur des méthodes apprenant un unique dictionnaire. Enfin, nous nous intéresserons plus particulièrement au schéma de codage développé autour de la structure appelée Iteration-Tuned and Aligned Dictionary (ITAD) [ZGK11, Zep10].

#### 1.3.2.1 Méthodes apprenant plusieurs dictionnaires

Les auteurs dans [BE08] proposent une méthode de compression d'images de visages spécifiquement, basée sur l'algorithme d'apprentissage de dictionnaire K-SVD [AEB06].

Une étape essentielle de la méthode est l'alignement géométrique des images d'apprentissage et de test, de telle façon que les caractéristiques principales des visages (oreilles, nez, bouche, ...) soient alignées pour toutes ces images. Plusieurs dictionnaires sont appris avec l'algorithme K-SVD sur des patchs prédéfinis correspondant à ces différentes caractéristiques de visage. Par exemple, un dictionnaire va être appris sur les patchs correspondant à l'œil gauche, un autre sur les patchs de la narine droite, etc..., de telle sorte que les dictionnaires soient particulièrement adaptés à une caractéristique du visage.

Chaque patch des images tests est ensuite décomposé par l'algorithme OMP [PRK93] sur le dictionnaire adéquat selon sa position, chaque patch étant associé à un des dictionnaires appris. La compression de chaque patch est alors effectuée en transmettant le vecteur parcimonieux de coefficients, c'est-à-dire les indices des coefficients non nuls et leurs valeurs.

Le nombre d'atomes alloué pour chaque patch est différent et dépend d'un seuil d'erreur de représentation à atteindre. Il est montré que davantage d'atomes sont attribués aux patchs contenant des détails du visage (cheveux, bouche, yeux et contours du visage). Cependant, même si ce nombre varie spatialement, il reste fixe pour chaque image grâce à l'alignement géométrique réalisé, de telle façon que le décodeur sache le nombre d'atomes alloués à chaque patch sans information supplémentaire à transmettre.

Les expérimentations montrent que cette méthode permet d'obtenir de meilleurs résultats (pouvant atteindre plusieurs dB) que les standards JPEG [Wal91] et JPEG 2000 [TM02] à bas débit, des résultats profitant de la méthode d'alignement géométrique appliquée aux images de visages.

Dans le cadre de la compression, une méthode est présentée dans [SHG08] permettant d'optimiser conjointement la classification des blocs et les transformées correspondantes sur un ensemble de vecteurs d'apprentissage. Les transformées classiquement utilisées en compression sont ici remplacées par des transformées orthonormales parcimonieuses. Ces transformées exploitent les orientations présentes dans l'image.

Le problème est donc de calculer les transformées orthonormales optimales $G_i \in \mathbb{R}^{n\times n}$ pour $k$ classes ($i = 1, ..., k$) de façon à minimiser le coût suivant sur les données d'entraînement :

$$\min_{G_i} \sum_{j\in S_i} \{\min_{x_i^j} ||y^j - G_i x_i^j||_2^2 + \lambda ||x_i^j||_0\} \text{ sous la contrainte } G_i^T G_i = I \tag{1.50}$$

où $S_i$ correspond à l'ensemble des blocs classifiés dans la classe $i$, $y^j \in \mathbb{R}^{N\times 1}$ le bloc $j$ des données d'apprentissage, $G_i$ la transformée orthonormale à optimiser et $x_i^j \in \mathbb{R}^{N\times 1}$ les coefficients de $y^j$ sur la transformée $G_i$.

De façon itérative, l'algorithme classifie chaque bloc des images et recherche la transformée optimale pour chaque classe, c'est-à-dire que $S_i$ et $G_i$ sont mis à jour alternativement de façon itérative.

Chaque bloc à encoder est ensuite représenté par la transformée optimale sur un critère débit-distorsion et les informations de classification sont encodées à l'aide d'un quadtree. Les coefficients sont quantifiés par un quantificateur scalaire à zone morte et codés de manière entropique.

Il est intéressant de noter que les transformées obtenues contiennent des structures directionnelles, bien que ces informations géométriques n'aient été utilisées qu'à l'initialisation des transformées. La méthode est donc particulièrement intéressante pour les images présentant des structures géométriques significatives et permet d'obtenir des résultats au niveau ou proches de ceux de JPEG 2000 à 0.5 bpp.

#### 1.3.2.2 Méthodes apprenant un unique dictionnaire

Les dictionnaires structurés en arbre [MV04] (voir section 1.2.8.1) ont également été intégrés dans un schéma de codage dans le contexte de la compression d'images de

visages [MJV04]. Un masque fixé à priori sur la zone des yeux, du nez et de la bouche permet de cibler la zone la plus pertinente.

L'image à encoder est décomposée en une composante basse fréquence et une composante haute fréquence. Un algorithme de poursuite basé arbre (comme [JVF06], voir section 1.1.3) est utilisé sur le dictionnaire arborescent afin d'encoder la composante haute fréquence de l'image. Les coefficients ainsi que les positions et indices des atomes sont ensuite codés. La procédure d'apprentissage permet de réduire la taille du dictionnaire et donc le nombre de bits nécessaires au codage des indices des atomes, accélérant de plus la procédure de codage et de décodage.

Cette méthode montre des gains en PSNR par rapport à JPEG 2000 à bas débit. A 0.3 bpp, des gains autour de 2.3 dB sont par exemple obtenus pour des images de visages hors de la base d'apprentissage.

Un schéma de compression s'adaptant à l'image d'entrée est proposé dans [HBR12]. En effet, l'image d'entrée est encodée en utilisant un dictionnaire spécialement appris sur cette image. Le dictionnaire doit donc être transmis au décodeur. C'est pourquoi un dictionnaire parcimonieux (Sparse K-SVD) [RZE10] est appris afin de profiter de sa représentation compacte. Ainsi, ce schéma de codage n'est pas adapté qu'à un type images (comme les images de visages) mais s'adapte au contraire à chaque image d'entrée.

Les patchs (sans chevauchement) de l'image d'entrée, auxquels leur valeur moyenne a été retirée, sont utilisés afin d'apprendre le dictionnaire parcimonieux $A$ avec l'algorithme Sparse K-SVD, en choisissant le dictionnaire sur-complet DCT comme dictionnaire de base $\Phi$. $A$ est ensuite quantifié et la décomposition parcimonieuse est calculée sur le dictionnaire quantifié $D_q = \Phi A_q$, avec une erreur cible globale sur l'image entière. Le dictionnaire parcimonieux $A$ et la matrice de coefficients $X$ sont quantifiés par une quantification uniforme puis sont ensuite codés de manière entropique avec un codeur arithmétique. Les valeurs moyennes des patchs sont également transmises.

Bien que le dictionnaire nécessite également d'être encodé et transmis, les résultats obtenus se situent globalement entre JPEG et JPEG 2000.

La méthode de codage d'image à bas débit présentée dans [FiVVF06] est basée sur du *matching pursuit* sur un dictionnaire redondant spécialement conçu pour représenter les caractéristiques 2D des images naturelles, en particulier les contours.

Le dictionnaire est construit en appliquant des transformations géométriques à une fonction de base. Chaque atome est ainsi indexé par des paramètres de translation, de *scaling* anisotrope et de rotation appliqués à la fonction de base, rendant le dictionnaire invariant à ces transformations. Les translations permettent de déplacer l'atome sur toute l'image, les rotations d'orienter l'atome localement le long d'un contour et le *scaling* anisotrope de s'adapter à la douceur du contour.

Les atomes construits par des fonctions Gaussiennes le long d'une direction et par la dérivée seconde de fonctions Gaussiennes dans la direction orthogonale permettent de capturer les contours de l'image. Les atomes formés par des fonctions Gaussiennes isotropes permettent quant à eux de représenter les basses fréquences de l'image.

Les coefficients de l'algorithme MP sont ensuite quantifiés, le nombre de niveaux de quantification étant calculé d'après une optimisation débit-distorsion. Puis un codage arithmétique adaptatif est appliqué aux coefficients quantifiés et aux paramètres des atomes sélectionnés.

Le schéma de codage décrit produit des résultats de reconstruction (en PSNR) similaires à JPEG 2000 à bas débit (parfois meilleurs à très bas débit). Cependant, l'écart entre les deux schémas de codage devient rapidement à l'avantage de JPEG 2000 lorsque le débit augmente, notamment à cause des limitations des transformées redondantes à haut débit. Le *matching pursuit* sur un dictionnaire redondant est ainsi particulièrement efficace à bas débit.

Visuellement, les artéfacts produits par le schéma de codage, ce dernier se concentrant sur les structures géométriques, sont moins gênants que ceux produits par les ondelettes dans JPEG 2000 à bas débit.

De plus, ce schéma de codage offre d'intéressantes possibilités d'adaptation, notamment en résolution spatiale ou en débit, et produit un *bitstream* flexible.

On notera également la contribution [SE11] explorant la capacité de compression de représentations parcimonieuses sur des dictionnaires appris via l'algorithme RLS-DLA (Recursive Least Squares Dictionary Learning Algorithm) [SE10], à la fois dans le domaine pixel et dans le domaine ondelette. Cet algorithme permet de mettre à jour le dictionnaire à chaque itération d'après un nouveau vecteur d'entraînement. La valeur moyenne des patchs est codée séparément via une méthode prédictive similaire à une DPCM (Differential Pulse-Code Modulation), suivie par un codage de Huffman des erreurs de prédiction. Les coefficients de la décomposition sur le dictionnaire appris sont ensuite quantifiés de manière uniforme avec seuillage. Les coefficients non nuls sont alors mis sous la forme d'une séquence et leurs informations de position, via le nombre de zéros précédant chaque coefficient non nul, dans une deuxième séquence. Un codage de Huffman est enfin appliqué aux deux séquences. Le RLS-DLA offre de meilleures performances que K-SVD notamment, pour la compression d'images génériques, et permet d'obtenir dans le domaine ondelette des résultats légèrement en dessous de ceux de JPEG 2000.

Enfin, récemment, une approche ajoutant une contrainte de compressibilité sur les coefficients parcimonieux a été proposée dans [XLLZ14] pour la compression d'images à bas débit. Les auteurs souhaitent utiliser la relaxation convexe, pour sa stabilité, en utilisant la norme $l_1$ plutôt que la norme $l_0$ pour mesurer la parcimonie. Mais la norme $l_1$ fournissant des coefficients moins compressibles que l'algorithme MP, une contrainte de compressibilité est ajoutée aux coefficients parcimonieux. Elle permet de s'affranchir des coefficients de petite valeur, pénalisant la compression à bas débit, et impose aux coefficients (rangés en ordre décroissant) de suivre une certaine loi de décroissance.

De plus, afin de réduire les effets de blocs propres au traitement de chaque bloc individuellement, les blocs se chevauchent horizontalement et verticalement, mais au prix d'un coût en débit-distorsion, plus de patchs devant être codés. Ainsi, une contrainte de consistance entre les patchs se chevauchant est également ajoutée au problème.

Cette approche CCSR (Compressibility Constrained Sparse Representation) permet donc de trouver un compromis entre erreur d'approximation des patchs, compatibilité des patchs voisins, et parcimonie et compressibilité des coefficients.

Une prédiction DPCM et un codage de Huffman sont appliqués afin de coder les valeurs moyennes des patchs de l'image. Une méthode K-means est utilisée pour la quantification des coefficients et un codage de Huffman est appliqué, spécifiquement pour l'image à coder. La table de Huffman est donc également encodée. Un code à longueur fixe est utilisé pour les indices ($\log_2(K)$, avec $K$ le nombre d'atomes du dictionnaire $D$).

Une méthode d'apprentissage de dictionnaire, basée sur un algorithme de descente de gradient, est de plus proposée. Celle-ci prend également en compte la contrainte de compressibilité des coefficients.

L'approche CCSR permet en général de surpasser RLS-DLA [SE11], le codeur basé MP [FiVVF06] et JPEG 2000 (+0.9 dB en moyenne sur des images naturelles et aériennes à 0.4 bpp) à bas débit, à la fois au niveau du PSNR et visuellement.

#### 1.3.2.3 Le codeur ITAD

Décrivons maintenant le codeur développé autour de la structure ITAD [ZGK11, Zep10], dont nous reprendrons le schéma dans la suite de cette thèse.

Le codec ITAD permet de compresser une image en décomposant chacun de ses blocs de façon parcimonieuse grâce au dictionnaire ITAD et en déterminant la parcimonie par bloc grâce à un critère débit-distorsion.

Pour chaque bloc carré, les blocs ne se chevauchant pas, la moyenne, que l'on nommera DC, est d'abord retirée. Les valeurs de moyenne sont codées séparément à l'aide d'une prédiction DPCM, les symboles obtenus étant par la suite encodés avec un codeur entropique.

Les blocs, dont la moyenne a été retirée, sont ensuite décomposés sur le dictionnaire ITAD qui a au préalable été appris sur un ensemble d'images d'apprentissage. Chaque bloc est ainsi transformé et représenté par un ensemble de paires indice-coefficient $(a_i, x_i)$, $a_i$ représentant l'indice de l'atome $d_i$ choisi au niveau $i$ au sein du dictionnaire $D_i$ et $x_i$ le coefficient associé.

Les coefficients sont d'abord quantifiés $(\tilde{x_i})$ par une quantification scalaire uniforme, et un codage entropique de Huffman, spécifique à chaque niveau de la structure, est ensuite appliqué. A noter que les tables de Huffman ont été préalablement construites hors ligne grâce aux images d'entraînement.

Les indices sont quant à eux codés avec un code à longueur fixe. En considérant que chaque dictionnaire $D_i$ est composé de $K$ atomes, le débit associé à chaque indice d'atome est :

$$R(a_i) = \log_2(K) \tag{1.51}$$

Le codec cherche à minimiser l'erreur de représentation de chaque bloc de l'image, sous la contrainte d'un débit maximum alloué à ne pas dépasser. Pour cela, la parcimonie par bloc diffère selon les blocs. Les approximations de tous les blocs sont initialisées à zéro. Puis, un bloc est sélectionné et son approximation est améliorée par l'ajout d'une

nouvelle paire $(a_i, \tilde{x_i})$ à sa représentation. La sélection des blocs est ainsi effectuée jusqu'à ce que le débit maximum alloué soit atteint. Le bloc choisi à chaque fois, afin d'en améliorer la représentation, est sélectionné selon un critère débit-distorsion. Le bloc choisi est en fait celui qui maximise un gain mesurant la diminution de l'erreur de reconstruction du bloc, due à l'ajout d'un nouvel atome dans la représentation, sur le coût en débit associé à l'ajout du couple $(a_i, \tilde{x_i})$ dans la décomposition. Le gain $G$ associé au couple $(a_i, \tilde{x_i})$ s'écrit donc :

$$G(a_i, \tilde{x_i}) = \frac{||y - \tilde{y}^{i-1}||_2^2 - ||y - \tilde{y}^{i}||_2^2}{R(a_i, \tilde{x_i})} \tag{1.52}$$

avec $\tilde{y}^i$ l'approximation du bloc $y$ au niveau $i$, c'est-à-dire avec $i$ atomes sélectionnés pour la représentation. Le débit associé à la paire $(a_i, \tilde{x_i})$, $R(a_i, \tilde{x_i})$, correspond au débit du coefficient quantifié égal à la longueur du code de Huffman associé, au débit de l'indice $R(a_i)$ et au débit de signalisation.

Enfin, le bitsream est structuré par bloc. Pour chaque bloc, il contient d'abord la valeur de la moyenne du bloc, puis les codes des différentes pairs indice-coefficient, séparés par une signalisation "EOB" (End Of Block) de 1 bit signalant si la paire juste codée est la dernière paire du bloc ou non.

Appliqué sur des images spécifiques de visages, ce codeur est capable de surpasser JPEG 2000 à bas débit (+0.9 dB à 0.4 bpp par exemple).

### 1.3.3 Classification supervisée

Hormis la compression, d'autres applications des dictionnaires appris, en rapport avec l'imagerie satellite, sont étudiées dans la troisième partie de cette thèse. L'une d'elle est la reconnaissance de scènes spécifiques au sein d'images satellites, qui peut être vue comme un problème de classification supervisée. Il existe dans la littérature de nombreuses méthodes traitant le problème de classification supervisée à l'aide de dictionnaires appris et des représentations parcimonieuses. Des méthodes utilisent également les représentations parcimonieuses sur des dictionnaires non appris, génériques ou composés de vecteurs d'entraînement (comme c'est le cas dans [WYG+09] avec une application à la reconnaissance faciale), mais nous nous concentrerons ici sur les méthodes utilisant l'apprentissage de dictionnaire pour le problème de classification supervisée. En considérant donc les labels des classes connus, un ou plusieurs dictionnaires sont appris, puis les représentations parcimonieuses sur ce ou ces dictionnaires sont utilisées afin de classifier des données de classe inconnue. Nous allons distinguer deux types de méthodes : les méthodes apprenant un dictionnaire unique, utilisant en général les coefficients des représentations pour la classification, et les méthodes apprenant un dictionnaire par classe, utilisant l'erreur de reconstruction sur les différents dictionnaires pour classifier. Une méthode à l'intersection de ces deux catégories sera également présentée.

#### 1.3.3.1 Méthodes apprenant un unique dictionnaire global

Commençons par les méthodes apprenant un dictionnaire unique.

Dans [RS08], une méthode d'apprentissage de dictionnaires permettant d'obtenir des représentations parcimonieuses simultanées et une classification robuste est présentée. L'apprentissage d'un unique dictionnaire utilisé pour toutes les classes est réalisé via une itération entre une étape de décomposition parcimonieuse opérée de façon simultanée par classe avec une contrainte de discrimination entre les classes, et une étape de mise à jour du dictionnaire par la méthode de SVD, de façon similaire à la mise à jour dans K-SVD. De cette façon, l'apprentissage intègre des termes de reconstruction et de discrimination.

Des représentations parcimonieuses simultanées sont recherchées pour chaque classe lors de la première étape, afin d'en extraire leur structure interne, tout en associant un terme de discrimination global entre les classes. Pour cela, l'algorithme de *Simultaneous Orthogonal Matching Pursuit* (SOMP) [TGS06] a été étendu. SOMP représente simultanément un ensemble de signaux comme une combinaison linéaire d'un sous-ensemble commun d'atomes du dictionnaire. Les signaux sont ainsi tous représentés par différentes combinaisons linéaires des mêmes atomes (seuls les coefficients diffèrent). Une mesure de discrimination sur les coefficients a été incorporée à l'algorithme SOMP et est globale, tandis que le terme de reconstruction est traité par classe. Le terme de discrimination sur les coefficients fait intervenir les matrices de dispersion (*scatter matrices* en anglais) intra-classes et inter-classes en cherchant à diminuer la dispersion des échantillons au sein d'une même classe et à l'augmenter entre les différentes classes. Les matrices de dispersion intra et inter-classes représentent respectivement, à des facteurs près, la moyenne sur toutes les classes des matrices de covariance de chaque classe, et la matrice de covariance des centroïdes des classes.

La classification est réalisée en utilisant des *Machines à Vecteurs Supports* (en anglais *Support Vector Machine*, SVM) linéaires sur les coefficients parcimonieux.

Ces travaux montrent l'importance de l'apprentissage de dictionnaires par rapport à des dictionnaires prédéfinis pour la classification, qui plus est des dictionnaires appris à la fois pour la reconstruction et discriminants afin d'obtenir une classification précise et robuste.

Un unique dictionnaire partagé par toutes les classes est également appris dans [MBP+08b], ainsi que des modèles associés à chaque classe, dans une formulation à la fois générative et discriminante, avec un modèle discriminant différent comparé à [RS08]. Le dictionnaire est ainsi appris de façon supervisée pour la classification : une étape de codage parcimonieux supervisé permet de calculer les coefficients parcimonieux et une étape d'apprentissage supervisé du dictionnaire permet de mettre à jour le dictionnaire ainsi que les paramètres du modèle du classifieur.

Deux modèles différents utilisant les coefficients parcimonieux sont utilisés pour la classification : un modèle linéaire et un modèle bilinéaire.

Le dictionnaire et les modèles appris sont alors utilisés dans le cadre de la reconnaissance de chiffres et de la classification de textures. L'intérêt de la discrimination du dictionnaire est alors montré. Sur des images de texture, le taux d'erreur de classification peut par exemple passer de 18.99% pour un dictionnaire non discriminant à 14.26% pour le dictionnaire discriminant, lorsque 30000 vecteurs sont utilisés pour

l'apprentissage.

Les auteurs proposent dans [ZL10] d'apprendre un dictionnaire avec une version discriminante de K-SVD [AEB06], dans le cadre de la reconnaissance faciale. Le dictionnaire appris doit ainsi posséder un bon pouvoir de représentation tout en offrant de la discrimination entre les classes. Pour cela, l'erreur de classification est ajoutée au sein du problème à optimiser qui devient :

$$< D, W, X >= \underset{D,W,X}{\arg\min} || \begin{pmatrix} Y \\ \sqrt{\gamma}H \end{pmatrix} - \begin{pmatrix} D \\ \sqrt{\gamma}W \end{pmatrix} X||_2 \text{ sous la contrainte } ||X||_0 \leq T \quad (1.53)$$

avec $Y$ l'ensemble des vecteurs d'apprentissage, $H$ leurs labels (chaque colonne $h_i = [0, ..., 0, 1, 0, ..., 0]$ contient une valeur non nulle dont la position indique la classe), $D$ le dictionnaire, $X$ la matrice de coefficients parcimonieux, $T$ un paramètre permettant d'imposer la parcimonie et $W$ le classifieur. $\gamma$ est un scalaire permettant de contrôler la contribution relative des différents termes.

Le problème est alors similaire à celui de K-SVD, et K-SVD permet d'en trouver la solution optimale. Plutôt que de résoudre alternativement des sous-problèmes de façon itérative pour approximer la solution, cette méthode permet ainsi de calculer directement et de façon simultanée le dictionnaire et le classifieur. Mais comme $D$ et $W$ sont normalisés de façon conjointe (c'est-à-dire $|| \begin{pmatrix} d_i \\ \sqrt{\gamma}w_i \end{pmatrix} ||_2 = 1$), le dictionnaire normalisé $D'$ et le classifieur correspondant $W'$ sont calculés de la façon suivante :

$$D' = \{\frac{d_1}{||d_1||_2}, \frac{d_2}{||d_2||_2}, ..., \frac{d_K}{||d_K||_2}\} \quad (1.54)$$

$$W' = \{\frac{w_1}{||d_1||_2}, \frac{w_2}{||d_2||_2}, ..., \frac{w_K}{||d_K||_2}\} \quad (1.55)$$

Une image test $y$ est alors décomposée de façon parcimonieuse sur $D'$ afin de déterminer les coefficients parcimonieux $x'$, par exemple avec l'algorithme OMP [PRK93]. La classification de cette image test est ensuite basée sur ses coefficients parcimonieux $x'$. Le classifieur $W'$ est simplement appliqué à $x'$ afin d'obtenir le label de l'image :

$$l = W'x' \quad (1.56)$$

$l$ étant un vecteur de $C$ (nombre de classes) composantes, le label est donné par l'index de $l$ correspondant à la valeur maximale contenue dans $l$.

Testée sur deux bases d'images de visages, cette version discriminante de K-SVD permet d'obtenir une meilleure classification que K-SVD (+2.4% et +6.8%).

Un algorithme supervisé, nommé *Label Consistent K-SVD* (LC-KSVD), permettant d'apprendre un dictionnaire discriminant est présenté dans [JLD11]. Au sein de ce dictionnaire, une information de label est associé à chaque atome (plusieurs atomes étant associés à chaque label) dans le but de renforcer la discrimination des coefficients

parcimonieux lors de l'apprentissage du dictionnaire. Pour cela, le problème à optimiser présenté dans [ZL10] est étendu et une nouvelle contrainte de consistance de label, mesurant une erreur discriminante des coefficients parcimonieux, est ajoutée à l'erreur de reconstruction et l'erreur de classification. Grâce à cette contrainte de consistance de label, lors de l'étape de classification, un atome du dictionnaire contribuera particulièrement à une classe de signaux, même si des signaux d'autres classes peuvent l'utiliser également. Le problème devient alors :

$$< D, W, A, X >= \underset{D,W,A,X}{\arg\min} ||Y - DX||_2^2 + \alpha ||Q - AX||_2^2 + \beta ||H - WX||_2^2 \text{ sous la contrainte } \forall i, ||x_i||_0 \leq T \tag{1.57}$$

Le premier terme représente l'erreur de reconstruction et le dernier l'erreur de classification, définis comme pour [ZL10] avec $W$ les paramètres du classifieur et $H \in \mathbb{R}^{C \times N}$ (avec $C$ le nombre de classes) les labels des signaux d'apprentissage $Y$, chaque colonne $h_i = [0, ..., 0, 1, 0, ..., 0] \in \mathbb{R}^C$ correspondant à un signal $y_i$ et où la valeur non nulle indique le label de $y_i$. Le terme $||Q - AX||_2^2$ représente l'erreur discriminante des coefficients parcimonieux incitant les coefficients dans $X$ à approximer les codes parcimonieux discriminants dans $Q$. $Q = [q_1 ... q_N] \in \mathbb{R}^{K \times N}$ représente les codes parcimonieux discriminants des signaux $Y$. Chaque colonne $q_i = [q_i^1 ... q_i^K]^t = [0...1, 1, ...0]^t \in \mathbb{R}^K$ est un code parcimonieux discriminant correspondant à un signal $y_i$ dont les valeurs non nulles sont placées aux indices $k$ où le signal $y_i$ et l'atome du dictionnaire $d_k$ possèdent le même label. Ainsi, ce terme force les signaux appartenant à la même classe à avoir des représentations parcimonieuses, c'est-à-dire des vecteurs de coefficients, similaires, ce qui est important pour l'étape de classification.

Le problème peut être réécrit ainsi :

$$< D, W, A, X >= \underset{D,W,A,X}{\arg\min} || \begin{pmatrix} Y \\ \sqrt{\alpha} Q \\ \sqrt{\beta} H \end{pmatrix} - \begin{pmatrix} D \\ \sqrt{\alpha} A \\ \sqrt{\beta} W \end{pmatrix} X ||_2^2 \text{ sous la contrainte } \forall i, ||x_i||_0 \leq T \tag{1.58}$$

Comme dans [ZL10], l'algorithme K-SVD est alors utilisé afin d'apprendre conjointement un dictionnaire discriminant compact et un classifieur linéaire. LC-KSVD apprend ainsi simultanément $D$, $A$ et $W$, ce qui permet d'éviter d'obtenir un minimum local comme solution.

De façon similaire à [ZL10], $D'$ et $W'$ sont calculés, ainsi que $A'$ :

$$A' = \{ \frac{a_1}{||d_1||_2}, \frac{a_2}{||d_2||_2}, ..., \frac{a_K}{||d_K||_2} \} \tag{1.59}$$

La classification d'une image test $y$ est réalisée de la même manière que [ZL10], en calculant tout d'abord ses coefficients parcimonieux $x'$ sur $D'$, puis en leur appliquant le classifieur $W'$ afin d'estimer le label correspondant à l'indice de la valeur maximale du vecteur $l = W'x'$.

La méthode est utilisée dans le cadre de la reconnaissance faciale et d'objets et présente de meilleurs résultats que K-SVD [AEB06] et sa version discriminante [ZL10].

L'algorithme d'apprentissage de dictionnaire en ligne [MBPS10] est utilisé dans [GGK11] pour apprendre encore une fois un unique dictionnaire sur l'ensemble des données d'apprentissage, quelque soit leur classe, c'est-à-dire de façon non supervisée. Cependant, aucune contrainte discriminante n'est ici ajoutée.

Suite à l'apprentissage, un modèle doit être trouvé pour chaque image de texture. Pour cela, des patchs de l'image sont décomposés de façon parcimonieuse sur le dictionnaire appris afin de calculer les coefficients parcimonieux, avec une contrainte de positivité, qui vont permettre de déterminer les atomes de $D$ contribuant à la représentation de chaque patch. En additionnant l'ensemble des coefficients des patchs de l'image, pour chaque atome, il est possible de calculer un histogramme de la contribution des atomes de $D$ pour la représentation de la texture particulière présente sur cette image. Cet histogramme sert alors de signature pour cette image de texture particulière.

Dans le cadre de la classification de texture, une image test de texture est ensuite classifiée en calculant son histogramme de la même façon et en cherchant, parmi les histogrammes calculés sur les images d'apprentissage, lequel est le plus proche.

#### 1.3.3.2 Méthode intermédiaire

Les auteurs présentent dans [YZF11] une méthode d'apprentissage d'un dictionnaire structuré dont les atomes ont une correspondance avec les labels des classes. Cette méthode se situe donc à l'intersection des deux catégories de méthodes présentées. De cette façon, en plus des coefficients, l'erreur de reconstruction après décomposition, associée à chacune des classes, peut également être utilisée comme critère pour la classification, contrairement aux méthodes précédentes apprenant un dictionnaire global dont les atomes ne présentent pas de correspondance avec les labels des classes (hormis LC-KSVD). Le processus d'apprentissage fait que chaque sous-dictionnaire associé à une classe, au sein du dictionnaire structuré, offre une bonne représentation aux signaux d'entrainements associés à cette classe, mais une mauvaise représentation aux signaux des autres classes. De plus, le critère de discrimination de Fisher est imposé aux coefficients parcimonieux afin de les rendre discriminants et qu'ils présentent une faible dispersion au sein d'une classe et une forte dispersion entre les classes. Pour cela, ce critère fait intervenir les matrices de dispersion intra-classes et inter-classes.

En définissant le dictionnaire $D = [D_1, D_2, ..., D_C]$ où chaque sous-dictionnaire $D_i$ est associé à la classe $i$ et $C$ est le nombre de classes, $Y = [Y_1, Y_2, ..., Y_C]$ l'ensemble des signaux d'entraînement ($Y_i$ étant le sous-ensemble de la classe $i$), et $X$ la matrice de coefficients telle que $X_i$ est la sous-matrice contenant les coefficients représentant $Y_i$ sur $D$, le modèle obtenu est alors le suivant :

$$J_{(D,X)} = \underset{(D,X)}{\arg\min}\{\sum_{i=1}^{C} r(Y_i, D, X_i) + \lambda_1 ||X||_1 + \lambda_2 f(X)\} \tag{1.60}$$

avec $r(Y_i, D, X_i)$ le terme de fidélité discriminant, $||X||_1$ la contrainte de parcimonie, $f(X)$ le terme, basé sur le critère de discrimination de Fisher, imposant une contrainte de discrimination aux coefficients en minimisant la dispersion de $X$ au sein d'une classe

et en la maximisant entre les classes, et $\lambda_1$ et $\lambda_2$ des paramètres scalaires. Le terme de fidélité discriminant est défini par :

$$r(Y_i, D, X_i) = ||Y_i - DX_i||_F^2 + ||Y_i - D_i X_i^i||_F^2 + \sum_{j=1, j\neq i}^{C} ||D_j X_i^j||_F^2 \tag{1.61}$$

avec $X_i^j$ les coefficients de $Y_i$ correspondant au sous-dictionnaire $D_j$. Cette expression permet que $Y_i$ soit bien représenté par le dictionnaire $D$ global, et en particulier par le sous-dictionnaire $D_i$, mais pas par les sous-dictionnaires $D_j, j \neq i$. Le problème (1.60) est résolu de façon itérative en optimisant alternativement le dictionnaire discriminant $D$ et les coefficients discriminants $X$.

Pour la classification, les coefficients sont calculés soit de manière globale par une décomposition sur le dictionnaire entier, soit de manière locale par une décomposition sur chaque sous-dictionnaire. La classification est alors réalisée en utilisant à la fois l'erreur de reconstruction par classe et les coefficients parcimonieux (via une distance avec le vecteur de coefficients moyen de la classe), les deux entités ayant été rendues discriminantes par la méthode d'apprentissage.

La méthode est appliquée à la reconnaissance faciale, de chiffres et de genres. Pour la reconnaissance de chiffres, la méthode obtient des résultats proches de ceux du dictionnaire discriminant dans [MBP+08b]. Pour les autres applications, elle surpasse plusieurs méthodes dont le dictionnaire K-SVD discriminant [ZL10].

#### 1.3.3.3 Méthodes apprenant un dictionnaire par classe

Intéressons nous désormais aux méthodes apprenant un dictionnaire par classe et utilisant l'erreur de reconstruction sur les différents dictionnaires pour classifier.

Dans [RSS10], un dictionnaire est appris par classe, de façon à représenter au mieux de façon parcimonieuse les signaux des différentes classes. Un terme d'incohérence encourage les dictionnaires à être aussi indépendants que possible entre eux, bien que les classes puissent tout de même partager certaines caractéristiques. Le problème est donc d'optimiser :

$$\min_{\{D_i, X_i\}_{i=1,...,C}} \sum_{i=1}^{C} \{||Y_i - D_i X_i||_2^2 + \lambda \sum_{j=1}^{N_i} ||x_i^j||_1\} + \eta \sum_{i\neq j} ||D_i^T D_j||_F^2 \tag{1.62}$$

avec $Y_i, i = 1, ..., C$ correspondant à la collection des $C$ classes de signaux d'apprentissage, $D_i$ les dictionnaires des différentes classes et $X_i = [x_i^1 ... x_i^{N_i}]$ la matrice de coefficients de la classe $i$. Le premier terme constitue le terme de reconstruction des signaux de la classe (avec la contrainte de parcimonie) et le second ajoute de l'incohérence entre les dictionnaires.

Afin de classifier un signal $y$ grâce aux dictionnaires appris, une mesure de qualité des représentations parcimonieuses $R$ sur les différents dictionnaires, faisant intervenir la norme $l_1$ sur les coefficients, est utilisée :

$$R(y, D_i) = \min_x ||y - D_i x||_2^2 + \lambda ||x||_1 \tag{1.63}$$

Cette mesure permet de prendre en compte l'erreur de reconstruction ainsi que la parcimonie de la représentation. Le signal est ainsi décomposé sur chaque dictionnaire et celui permettant d'obtenir la meilleure représentation parcimonieuse, c'est-à-dire minimisant $R$, indique la classe. A noter que les coefficients associés à des atomes communs entre les différentes classes sont ignorés pour le calcul de $R$, afin d'améliorer la discrimination.

Cette méthode est appliquée à une tâche de détection d'objets et est par la suite étendue à un problème de classification non supervisée, avec un algorithme itératif alternant entre une étape assignant chaque donnée au cluster pour lequel la meilleure représentation est obtenue et une étape d'apprentissage des dictionnaires sur les clusters calculés à l'étape précédente.

Le terme d'incohérence entre les dictionnaires est également utilisé dans [KW12]. Les auteurs y proposent une méthode permettant de séparer les éléments communs aux dictionnaires des différentes classes afin de ne conserver dans ces dictionnaires que leurs spécificités vis-à-vis de leur classe. Ainsi, un dictionnaire spécifique par classe est appris afin d'apprendre les particularités de chaque classe, et un autre dictionnaire est simultanément appris afin d'apprendre les éléments communs aux différentes classes. $C + 1$ dictionnaires sont donc appris, avec $C$ le nombre de classes. Cela permet de rendre les dictionnaires associés aux différentes classes plus discriminants.

Pour cela, la fonction à minimiser intègre différents termes avec différents objectifs : un terme mesurant l'erreur de reconstruction des données de chaque classe sur le dictionnaire entier (composé de la concaténation des $C + 1$ dictionnaires) avec une contrainte sur la parcimonie des coefficients ; un terme mesurant l'erreur de reconstruction des données de chaque classe en utilisant uniquement le dictionnaire de la classe correspondante $D_c$ et le dictionnaire commun $D_{C+1}$ ; un terme forçant les coefficients sur les autres dictionnaires (autres que $D_c$ et $D_{C+1}$) à être faibles afin d'accroitre la discrimination des dictionnaires ; et un terme d'incohérence entre les dictionnaires afin de différencier les dictionnaires entre eux, et notamment que les éléments communs aux différentes classes se retrouvent bien dans le dictionnaire commun $D_{C+1}$ uniquement.

Plusieurs méthodes de classification sont proposées, dont une globale et une locale selon que la décomposition soit effectuée respectivement sur le dictionnaire entier ou sur chaque dictionnaire de classe $D_c$ (concaténé avec le dictionnaire commun $D_{C+1}$). L'erreur de reconstruction, calculée en utilisant le dictionnaire spécifique $D_c$ et le dictionnaire commun $D_{C+1}$ pour chaque classe $c$, est ensuite utilisée comme critère de classification, en cherchant la classe la minimisant.

Cette méthode est appliquée à diverses applications telles que la reconnaissance faciale, de chiffres, d'objets ou bien la classification de scène.

Enfin, dans [MBP+08a], les auteurs s'intéressent à de l'analyse locale d'image en travaillant par patchs. Des dictionnaires à la fois discriminants et pour la reconstruction (un pour chaque classe) sont appris simultanément grâce à une méthode optimisant conjointement un critère de reconstruction parcimonieuse et un critère de discrimination entre les différentes classes. Les dictionnaires ainsi appris sont alors efficaces pour

représenter les données de la classe à laquelle ils correspondent et discriminants par rapport aux données des autres classes.

Les erreurs de reconstruction sur ces multiples dictionnaires de patchs d'images sont calculées afin d'aboutir à une classification au niveau pixel. Pour cela, les coefficients parcimonieux sont d'abord calculés pour un vecteur $y$ sur chaque dictionnaire $D_i$ avec la contrainte de parcimonie $L$ :

$$x^\star(y, D_i) = \arg\min_x ||y - D_i x||_2^2 \text{ sous la contrainte } ||x||_0 \leq L \tag{1.64}$$

Les erreurs de représentations correspondantes sont alors :

$$R^\star(y, D_i) = ||y - D_i x^\star(y, D_i)||_2^2 \tag{1.65}$$

Le label $\hat{i}_0$ du signal $y$ correspond alors au dictionnaire $D_i$ offrant la meilleure représentation, c'est-à-dire l'erreur de représentation minimale :

$$\hat{i}_0 = \arg\min_{i=1,...,C} R^\star(y, D_i) \tag{1.66}$$

avec $C$ le nombre de classes.

Pour un apprentissage de dictionnaires à la fois efficaces pour la reconstruction et discriminants, le problème à résoudre est le suivant :

$$\min_{\{D_j\}_{j=1}^C} \sum_{i=1,...,C, l \in S_i} C_i^\lambda(\{R^\star(y_l, D_j)\}_{j=1}^C) + \lambda\gamma R^\star(y_l, D_i) \tag{1.67}$$

où le paramètre $\gamma$ contrôle le compromis entre la reconstruction (second terme) et la discrimination (premier terme), et $S_i$ correspond à la classe $i$. Le terme discriminant est défini en utilisant une fonction de coût *softmax* discriminante :

$$C_i^\lambda(r_1, r_2, ..., r_C) = \log(\sum_{j=1}^{C} \exp^{-\lambda(r_j - r_i)}) \tag{1.68}$$

qui est proche de zéro lorsque $r_i$ est la plus faible valeur parmi les $r_j$ et représente un coût de pénalité dans le cas contraire. Résoudre ce problème permet non seulement de rendre chaque dictionnaire $D_i$ efficace pour représenter les éléments de sa propre classe $S_i$, mais également d'en faire le meilleur dictionnaire pour $S_i$ parmi les $C$ dictionnaires appris simultanément, rendant l'erreur de représentation discriminante pour la classification.

Ce problème est résolu par un schéma itératif alternant entre codage parcimonieux sur les différents dictionnaires et mise à jour de tous les dictionnaires dans une approche présentant des similitudes avec l'algorithme K-SVD. Cet algorithme peut donc être vu comme une version discriminante de K-SVD.

Cette approche est notamment illustrée dans le cadre de la segmentation de texture et permet de montrer le gain apporté par l'approche discriminante par rapport à une approche de reconstruction pure.

## 1.4 Conclusion

Dans ce premier chapitre, nous avons introduit et exploré les concepts de représentations parcimonieuses et d'apprentissage de dictionnaires, devenus particulièrement usités ces derniers temps. Cette thèse portant sur le traitement d'images spécifiques que sont les images satellites, nous pensons qu'il est intéressant d'explorer ces domaines de recherche dans différents cadres tels que la compression ou encore la classification. Nous essayerons ainsi de voir si ces images satellites présentent assez de spécificités pour pleinement tirer partie de ces techniques d'apprentissage.

Nous avons de plus présenté différentes applications de ces méthodes : le débruitage, la compression et la classification supervisée. La compression d'images satellites sera l'objet de la deuxième partie de cette thèse. Pour cela, nous allons étudier des dictionnaires structurés dans la lignée des ITD, adaptés au codage. Nous explorerons ainsi différentes structures de dictionnaires afin d'améliorer leurs performances en représentation et en codage. Pour le codage, un schéma de compression similaire à celui d'ITAD sera utilisé. Il sera comparé à quelques standards de compression d'images fixes qui sont présentés dans le prochain chapitre.

La classification supervisée sera quant à elle abordée dans la troisième partie de cette thèse. Nous testerons alors le pouvoir de classification des dictionnaires structurés en apprenant un dictionnaire par classe et en utilisant l'erreur de reconstruction pour la classification, dans la lignée des travaux [RSS10] et [MBP+08a]. Une méthode similaire sera appliquée dans le cadre de la reconnaissance de FTM (Fonction de Transfert de Modulation), une application également étudiée dans la troisième partie de la thèse.

# Chapitre 2

# Quelques standards de compression d'images fixes

Dans ce chapitre, nous allons présenter les principaux concepts de plusieurs standards de compression d'images. Nous nous intéresserons tout d'abord à des codeurs spécialement dédiés au codage d'images fixes : JPEG, JPEG 2000 et le codeur CCSDS 122, standard développé pour le codage d'images spatiales. Puis nous nous intéresserons au cas de codeurs vidéos en mode Intra, mode utilisé pour coder les images de type $I$ au sein de vidéos, des images codées sans aucune prédiction temporelle par rapport à d'autres images de la séquence. En mode Intra, un codeur vidéo compresse ainsi chaque image indépendamment des autres. Appliqué à une image, cela revient à un codeur d'image fixe. Nous décrirons alors les blocs importants des codeurs H.264/AVC et HEVC (en mode Intra). Ces codeurs seront utilisés comme références de comparaison dans la suite de cette thèse pour les tests de compression.

## 2.1 Codeurs d'images fixes

Les codeurs d'images fixes servent exclusivement à encoder des images. Nous allons d'abord décrire deux exemples parmi les plus utilisés : JPEG et JPEG 2000, le premier étant basé sur une transformée DCT et le second sur une transformée en ondelettes. Puis nous nous attarderons sur le codeur CCSDS 122, un standard spécifique à l'imagerie spatiale, proche de JPEG 2000 dans son fonctionnement.

### 2.1.1 JPEG

JPEG [Wal92] est un standard de compression d'images fixes, en niveaux de gris et couleurs, développé par le *Joint Photographic Experts Group* (JPEG) et datant de 1992 (date d'adoption du standard international).

La transformation DCT, la quantification et le codage entropique sont les trois étapes clés de l'encodage JPEG (Fig. 2.1). Nous nous concentrerons dans ce paragraphe sur la compression d'une image en niveaux de gris, c'est-à-dire à une seule composante.

A noter également que nous nous limiterons à décrire le codec séquentiel dit "Baseline" de JPEG, la méthode JPEG la plus couramment implémentée.

La première étape de la méthode JPEG est de découper l'image source en blocs de taille $8 \times 8$.

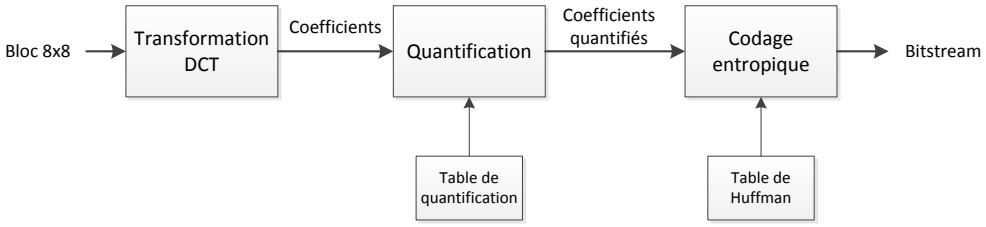



FIGURE 2.1 – Schéma d'un encodeur JPEG.

#### 2.1.1.1 La transformation DCT

A l'encodeur, les valeurs du bloc 8x8 sont d'abord décalées afin d'être centrées en 0. Puis le bloc est transformé en appliquant une transformée DCT [ANR74], correspondant à une DCT 2D de type II. La DCT calcule à l'encodeur un coefficient DCT $F(u,v)$ à partir d'une valeur $f(x,y)$ de la façon suivante :

$$F(u,v) = \frac{1}{4}C(u)C(v)[\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y)cos\frac{(2x+1)u\pi}{16}cos\frac{(2y+1)v\pi}{16}] \tag{2.1}$$

avec

$$\begin{aligned} C(u), C(v) &= \frac{1}{\sqrt{2}}\ pour\ u,v = 0 \\ C(u), C(v) &= 1\ autrement \end{aligned} \tag{2.2}$$

La transformée DCT permet de transformer les données originales dans un domaine fréquentiel dans lequel elles peuvent être plus facilement encodées. Les coefficients sont ainsi placés dans le bloc selon leur fréquence en $x$ et en $y$ suite à la transformée. Par convention, le coefficient dit DC, correspondant à des fréquences nulles en $x$ et en $y$, se situe en haut à gauche du bloc (Fig. 2.2). Plus on s'en éloigne, plus les coefficients dits AC correspondent à des coefficients de hautes fréquences. Étant donné que la valeur des pixels a tendance à varier doucement au sein d'une image, la plupart du signal est concentré dans les basses fréquences, ce qui est intéressant pour la compression. Pour un bloc 8x8 typique, la plupart des fréquences spatiales ont une amplitude nulle ou proche de 0 et n'ont donc pas besoin d'être encodées.

Au décodeur, la transformée DCT inverse est appliquée aux coefficients décodés afin de retrouver les valeurs du bloc 8x8 :

$$f(x,y) = \frac{1}{4}[\sum_{u=0}^{7}\sum_{v=0}^{7} C(u)C(v)F(u,v)cos\frac{(2x+1)u\pi}{16}cos\frac{(2y+1)v\pi}{16}] \tag{2.3}$$

#### 2.1.1.2 La quantification

A l'encodeur, après la transformée, chaque coefficient est quantifié à l'aide d'une table de quantification composée de 64 éléments. Ces éléments permettent de contrôler le ratio de compression.

L'étape de quantification permet d'obtenir une plus grande compression en quantifiant à 0 les faibles coefficients dont l'apport visuel ne serait pas significatif. Cette étape constitue la principale source de perte d'information du codeur.

Classiquement, les hautes fréquences étant moins perceptibles à l'œil, elles sont quantifiées avec un pas de quantification plus grand afin de retrouver de nombreux zéros parmi les coefficients de haute fréquence après quantification.

La quantification est définie par une division de chaque coefficient DCT par le pas de quantification correspondant dans la table de quantification, suivie par un arrondi à l'entier le plus proche.

La quantification inverse réalisée au décodeur consiste donc simplement en une multiplication par le pas de quantification correspondant dans la table de quantification.

#### 2.1.1.3 Le codage entropique

Après l'étape de quantification, l'étape finale est le codage entropique, apportant de la compression supplémentaire sans perte en encodant les coefficients DCT quantifiés d'après leurs caractéristiques statistiques.

Pour cela, le coefficient DC est traité différemment des coefficients AC. Étant donné qu'il y a souvent une forte corrélation entre les coefficients DC de deux blocs voisins, les coefficients DC quantifiés sont encodés par leur différence avec le DC du bloc précédent.

Quant aux coefficients AC quantifiés, ils sont d'abord ordonnés dans une séquence suivant un parcours en "zig-zag" (Fig. 2.2). Le but est ici de regrouper et placer les coefficients de basse fréquence avant ceux de haute fréquence, ces derniers contenant potentiellement beaucoup de coefficients nuls après la quantification.

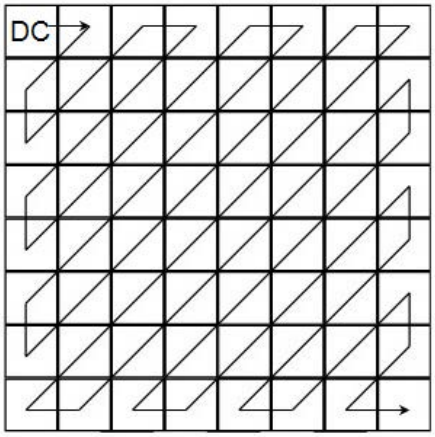



FIGURE 2.2 – Parcours en "zig-zag" des coefficients.

Le codage entropique est ensuite effectué en deux étapes : une première étape convertissant les coefficients DCT quantifiés en une séquence de symboles, puis une seconde étape assignant des codes de tailles variables aux symboles.

Un codage de type RLE (*Run Length Encoding*) est appliqué aux coefficients AC. Ainsi, chaque coefficient AC non nul est représenté en association avec le nombre de coefficients AC nuls qui le précèdent dans la séquence ordonnée en zig-zag. Chaque coefficient AC non nul est alors représenté par deux symboles. Le premier symbole est constitué de l'information *runlength* correspondant au nombre de valeurs 0 précédant le coefficient, et de l'information *size*, donnant le nombre de bits du second symbole. Ce second symbole contient simplement l'information d'*amplitude* correspondant à l'amplitude du coefficient non nul. Pour les coefficients DC, la représentation de la différence de DC avec le bloc précédent est similaire, mise à part le fait que le premier symbole comporte uniquement l'information *size*.

La seconde étape consiste ensuite à représenter les symboles par des codes à longueurs variables. Ainsi, les premiers symboles des coefficients DC et AC sont codés via des codes de Huffman [Huf52], de longueurs variables, contenus dans une table de Huffman. Cette table de Huffman contient également un code 'EOB' (*End Of Block*) permettant d'indiquer la fin d'un bloc, c'est-à-dire qu'il n'y a plus de coefficients à coder pour le bloc courant. Les deuxièmes symboles contenant l'information d'*amplitude* sont quant à eux représentés par des codes entiers à longueurs variables, mais qui ne sont pas des codes de Huffman, et correspondent à une représentation en bits du coefficient quantifié.

### 2.1.2 JPEG 2000

JPEG 2000 est un standard de compression d'images fixes [MGBB00, SCE01, TM02], avec ou sans perte, et datant de 2000 (draft final de la Partie 1 du standard international). Nous nous intéresserons ici à la Partie 1 du standard, correspondant au cœur de système de codage. Les étapes principales du schéma de codage JPEG 2000 sont présentées dans la figure 2.3 et sont décrites par la suite. Après divers prétraitements, le codage est réalisé par une transformée en ondelettes, suivie d'une étape de quantification et de codage entropique. A noter que ce standard surpasse JPEG du point de vue des performances.

#### 2.1.2.1 Étape de pré-traitements

La première étape consiste à découper l'image en "tuiles" (*tiles* en anglais), c'est-à-dire en blocs rectangulaires ne se chevauchant pas, afin qu'elles soient compressées de façon indépendante. Ainsi, toutes les opérations, incluant la transformation en ondelettes, la quantification et le codage entropique, sont réalisées de façon indépendante sur chaque tuile de l'image. Toutes les tuiles sont de même taille (excepté peut-être sur les bords), cette taille pouvant aller jusqu'à la taille de l'image elle-même, l'image étant alors considérée comme une seule tuile. Le fait de découper l'image en tuiles et de les traiter de façon indépendante permet de réduire les besoins en mémoire de l'algorithme,

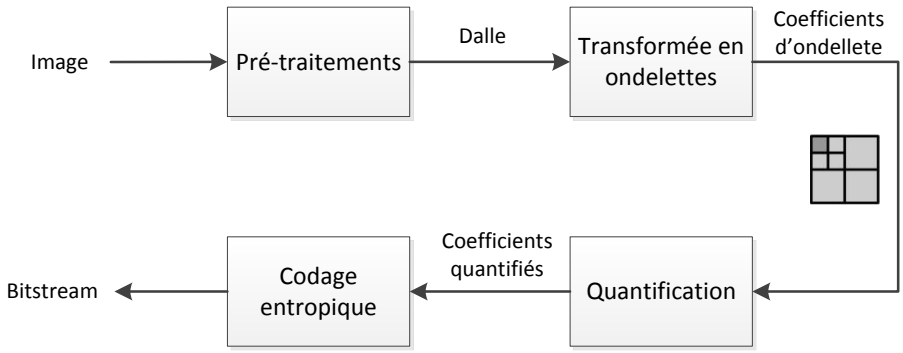



FIGURE 2.3 – Schéma d'un encodeur JPEG 2000.

mais au prix d'une baisse de performance car diviser en de nombreuses tuiles peut créer des effets de blocs.

Un décalage de la moyenne des valeurs est ensuite opéré, afin de les centrer en zéro, en soustrayant à chaque valeur la même quantité $2^{P-1}$, où $P$ représente la précision de chaque composante.

Enfin, dans le cas où une image couleur RVB (Rouge, Vert, Bleu) est compressée, une transformation de l'espace de couleur peut être appliquée. Ce nouvel espace de couleur permet de séparer la composante de luminance des deux composantes de chrominance, et est donc plus adaptée à la compression. Deux transformations des composantes couleur sont supportées par le standard : une transformation des composantes irréversible (ICT pour *Irreversible Component Transformation*), pouvant être utilisée pour le codage avec pertes, et une transformation des composantes réversible (RCT pour *Reversible Component Transformation*), pouvant être utilisée pour le codage avec ou sans pertes.

#### 2.1.2.2 La transformée en ondelettes

La prochaine étape consiste à appliquer une transformation en ondelettes sur les composantes de la tuile afin de les représenter avec différents niveaux de décompositions. Ces niveaux de décompositions contiennent des sous-bandes composées de coefficients décrivant les caractéristiques fréquentielles horizontales et verticales des valeurs originales de la tuile.

A chaque niveau de décomposition, une transformée en ondelettes en deux dimensions (2D) est appliquée en la séparant en deux transformations en une dimension (1D). Ainsi, la transformation 1D est d'abord appliquée aux lignes de l'image via un filtre d'analyse, passe-bas ($h_L$) ou passe-haut ($h_H$), suivie d'un sous-échantillonnage de facteur deux. Puis les mêmes filtres sont appliqués aux colonnes des données obtenues, avec un sous-échantillonnage de facteur deux. A chaque niveau de décomposition, une sous-bande de basses fréquences horizontales et verticales (BB), une sous-bande de hautes fréquences horizontales et de basses fréquences verticales (HB), une sous-bande

de basses fréquences horizontales et de hautes fréquences verticales (BH) et une sous-bande de hautes fréquences horizontales et verticales (HH) sont ainsi produites (Fig. 2.4).

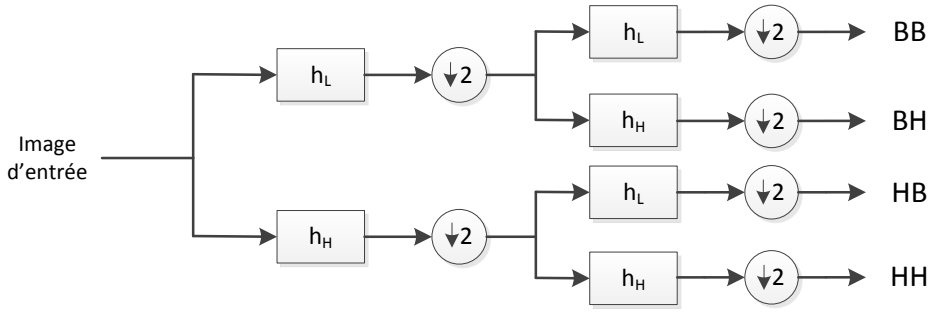



FIGURE 2.4 – Décomposition en ondelettes 2D.

Une décomposition dyadique de l'image sur plusieurs niveaux est réalisée en sous-bandes de fréquences en décomposant à chaque niveau la sous-bande BB en quatre sous-bandes plus petites (Fig. 2.5). L'image est finalement décomposée en $3N+1$ sous-bandes, avec $N$ le nombre de niveaux de décompositions appliqué.

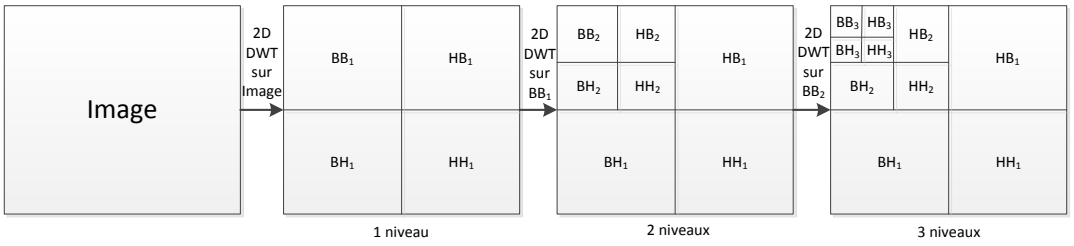



FIGURE 2.5 – Décomposition dyadique en ondelettes sur 3 niveaux.

Deux types de filtres peuvent être utilisés pour la transformation, correspondant à une transformation en ondelettes irréversible ou réversible. Le premier est le filtre de Daubechies 9/7 [CDF92, ABMD92]. Il est irréversible et définit un filtre passe-bas ($h_L$) de neuf coefficients et un filtre passe-haut ($h_H$) de sept coefficients. Le second est le filtre de Le Gall 5/3 [LGT88], qui est lui réversible et définit un filtre passe-bas ($h_L$) de cinq coefficients et un filtre passe-haut ($h_H$) de trois coefficients. A noter que le filtre de Le Gall 5/3, n'apportant pas de pertes, est associé à la transformation réversible des composantes couleurs (RCT) pour du codage avec ou sans perte ; tandis que le filtre Daubechies 9/7, apportant des pertes, est associé à la transformation irréversible des composantes couleurs (ICT) pour du codage avec pertes. L'ondelette 9/7 offre de meilleures performances que l'ondelette 5/3, mais au prix d'une complexité plus importante. Deux modes de filtrage sont supportés par le standard, basés convolution ou basé *lifting*.

#### 2.1.2.3 La quantification

Après la transformation, les coefficients sont quantifiés. Une quantification scalaire uniforme à zone morte est appliquée. Chaque coefficient $a_b(u, v)$ de la sous-bande $b$ est quantifié à la valeur $q_b(u, v)$ par la formule suivante :

$$q_b(u, v) = sign(a_b(u, v))\lfloor \frac{|a_b(u, v)|}{\Delta_b} \rfloor \tag{2.4}$$

Le pas de quantification $\Delta_b$ peut être différent pour chaque sous-bande $b$, mais un seul pas est permis par sous-bande. Ce pas influence la qualité et la compression : plus il est grand et plus la compression est importante et la perte de qualité est grande. La zone morte permet de mettre les faibles coefficients à zéro, les coder pouvant coûter cher pour un gain en qualité peu visible.

La quantification réduit ainsi la précision des coefficients et est donc responsable de pertes. Cependant, dans le cas de la compression sans perte, l'ondelette 5/3 produit des coefficients entiers et le pas de quantification appliqué est alors de 1, de façon à éviter toute perte d'information.

#### 2.1.2.4 Le codage entropique

Suite à la quantification, chaque sous-bande est divisée en blocs rectangulaires sans chevauchement. Trois rectangles correspondant à la même zone de l'image sur chaque sous-bande à un niveau de décomposition donné forment un "emplacement de partition de paquet", nommé *precinct*. Chaque precinct est ensuite divisé en rectangles réguliers sans chevauchement, nommés codes-blocs. Les codes-blocs, typiquement de taille 64x64, constituent l'entité fondamentale pour le codage entropique.

Le codage entropique est réalisé indépendamment sur chaque code-bloc, selon l'algorithme *Embedded Block Coding with Optimised Truncation* (EBCOT) [Tau00]. Les codes-blocs sont codés par plan de bits, en commençant par le plan de bits le plus significatif jusqu'au moins significatif. Chaque plan de bits est encodé en trois passes (on parle de sous plans de bits pour chaque passe), suivant un parcours particulier en colonnes parmi des bandes dont la hauteur est de quatre bits. Chaque bit est codé par une de ces trois passes, selon la signifiance de l'emplacement de ce bit et la signifiance des emplacements voisins. Un emplacement est considéré comme signifiant si un 1 a été codé à cet emplacement dans le plan de bit courant ou un plan de bit précédent.

La première passe est la passe de propagation de la signifiance (*significance propagation*). Un bit est codé lors de cette passe si son emplacement n'est pas signifiant, mais au moins un emplacement voisin (parmi les 8 emplacements voisins) est signifiant. Si un bit dont la valeur est 1 est codé lors de cette passe, alors son emplacement devient signifiant. La deuxième passe consiste en un affinage de l'amplitude (*magnitude refinement*). Durant cette passe, tous les bits d'emplacements devenus signifiants à un plan de bit précédent sont codés. Les emplacements devenus signifiants lors de la passe précédente du même plan de bit ne sont donc pas concernés. La troisième passe est la passe de nettoyage (*clean-up*), durant laquelle tous les bits non encodés lors des deux

premières passes le sont. A noter que le signe du coefficient est codé au plan de bit correspondant au premier bit à 1 codé.

Le codage est réalisé par un codeur arithmétique adaptatif, le codeur MQ, codant chaque bit selon son contexte, dépendant de la signifiance de son emplacement et des emplacements voisins

Pour chaque code-bloc, un *bitstream*, correspondant à un certain nombre de sous plans de bits, est généré. Une optimisation débit-distorsion permet de définir des points de troncature parmi chaque code-bloc. Les *bitstreams* des codes-blocs appartenant à un *precinct* forment un paquet. Puis en regroupant les paquets, correspondant aux différents *precincts* des différents niveaux de résolution, on obtient une couche. Un paquet apporte donc une amélioration de la qualité à un niveau de résolution donné et pour une certaine localisation spatiale, les *precincts* correspondant à des localisations spatiales. Une couche apporte quant à elle une amélioration de la qualité de l'image à pleine résolution. Le *bitstream* final est organisé comme une succession de couches.

En ordonnant les paquets différemment au sein du *bitstream*, différentes progressions peuvent être obtenues. Quatre sont possibles avec JPEG 2000 : une progression en résolution, en qualité, par emplacement spatial ou par composante. De plus, il n'y a pas de restriction quant au nombre de sous plans de bits de chaque code-bloc dans un paquet donné. Un encodeur peut ainsi formater les paquets selon l'application souhaitée.

Suite à la compression de l'image entière, une étape de post-traitement parcourt tous les codes-blocs et détermine la troncature à appliquer à chaque *bitstream* de code-bloc afin d'atteindre un débit ou une qualité cible.

#### 2.1.2.5 Quelques caractéristiques de JPEG 2000

Le standard JPEG 2000 présente certaines caractéristiques intéressantes, en particulier sa scalabilité spatiale et en qualité, ou encore la possibilité de définir une région d'intérêt dans l'image (ROI pour *Region Of Interest*).

Une propriété utile du standard est sa scalabilité en qualité et en résolution (scalabilité spatiale), c'est-à-dire sa capacité à réaliser le codage pour plusieurs qualités ou plusieurs résolutions simultanément. La scalabilité en qualité permet de générer progressivement plusieurs images à la même résolution spatiale mais à différentes qualités, à partir d'une unique image source. Tandis que la scalabilité en résolution permet de générer plusieurs images à différentes résolutions spatiales, à partir d'une unique image source.

Une autre caractéristique est la possibilité de définir une ROI, importante pour des applications où des parties de l'image sont de plus haute importance que d'autres. Ces régions doivent alors être encodées avec une meilleure qualité que l'arrière-plan. Dans le cas d'une transmission progressive du *bitstream*, ces régions doivent être transmises d'abord ou avec une forte priorité. Afin de coder la ROI, la méthode MAXSHIFT est utilisée au sein du standard JPEG 2000. Elle correspond à une extension de la méthode de codage de la ROI basée décalage. Le principe de cette méthode est de multiplier par un facteur les coefficients associés à la ROI afin que leurs bits soient décalés et placés dans des plans de bits plus hauts que ceux associés à l'arrière-plan.

De cette façon, la ROI est encodée et décodée avant le reste de l'image. La méthode MAXSHIFT permet de calculer le facteur à appliquer aux coefficients de telle façon que le coefficient minimum associé à la ROI soit supérieur au coefficient maximum associé à l'arrière-plan. La forme de la ROI n'a donc pas à être transmise.

### 2.1.3 CCSDS 122 : un standard pour l'imagerie satellite

Le standard CCSDS 122.0-B-1 [fSDS05] est un standard pour la compression de données image émanant du *Management Council of the Consultative Committee for Space Data Systems* (CCSDS) et datant de 2005. Il est recommandé pour les systèmes traitant des images spatiales et représente donc un standard pour la compression des images satellite. La technique de compression décrite dans ce standard peut être utilisée pour de la compression avec ou sans perte.

Ce standard présente certaines similitudes avec JPEG 2000. Cependant, il en est différent par les aspects suivants :

- Il cible spécialement les instruments à haut débit utilisés à bord d'engins spatiaux
- Un compromis est appliqué entre les performances de compression et la complexité
- La complexité moindre de ce standard permet des implémentations hardware rapides et à faible consommation
- Le nombre d'options est limité, permettant de l'utiliser sans une connaissance approfondie de l'algorithme.

Le codeur est constitué de deux blocs principaux (Fig. 2.6) : une transformée en ondelettes discrète responsable de la décorrélation des données et un encodeur par plans de bits encodant les données décorrélées.


FIGURE 2.6 – Schéma d'un encodeur CCSDS 122.

#### 2.1.3.1 La transformée en ondelettes

Une transformée en ondelette discrète (DWT pour *Discrete Wavelet Transform*) 2D séparable sur trois niveaux de décomposition est appliquée. Pour cela, l'utilisation d'une transformée 1D est répétée sur les lignes puis sur les colonnes de l'image d'entrée afin de la décomposer en quatre sous-bandes (Fig. 2.4). Puis à chaque niveau de décomposition, ce processus est appliqué à la sous-bande BB (Fig. 2.5).

Deux ondelettes 1D sont spécifiées dans le standard : la transformée en ondelettes discrète 9/7 biorthogonale, nommée "DWT flottante", et une approximation entière et non-linéaire de cette transformée, nommée "DWT entière". La "DWT flottante" donne en général de meilleurs résultats de compression avec pertes que la "DWT entière".

Cependant, seule cette dernière peut être utilisée dans le cas de la compression sans perte.

#### 2.1.3.2 L'encodeur par plans de bits

Suite à la transformée, les coefficients sont soit arrondis à l'entier le plus proche si la "DWT flottante" a été utilisée, soit multipliés par un facteur défini dans la norme (différent selon les sous-bandes) si la "DWT entière" a été utilisée.

Le codage des coefficients est réalisé par segment, chaque segment étant codé indépendamment. Un segment est constitué de $S$ blocs consécutifs. Chaque bloc est constitué de 64 coefficients provenant des différentes sous-bandes (Fig. 2.7) et correspond à un certain emplacement spatial de l'image originale. Parmi les 64 coefficients, on retrouve un coefficient DC dans la sous-bande $BB_3$ et 63 coefficients AC provenant des autres sous-bandes : 3 des autres sous-bandes du niveau 3 (1 par sous-bande), 12 des sous-bandes de niveau 2 (4 par sous-bande) et 48 des sous-bandes de niveau 1 (16 par sous-bande). Les blocs sont traités par l'encodeur et rangés dans les segments dans l'ordre dans lequel les coefficients DC correspondant sont rangés dans la sous-bande $BB_3$, ligne après ligne et de gauche à droite.

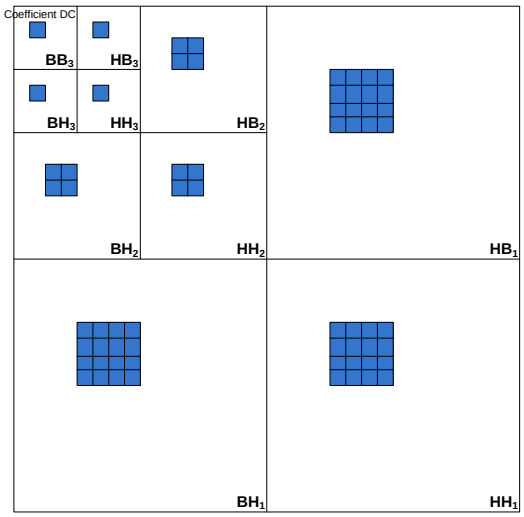



FIGURE 2.7 – Schéma d'un bloc (en bleu) au sein de l'image transformée

Les coefficients DC sont représentés en complément à deux et les coefficients AC dans une représentation binaire de leur signe puis de leur magnitude. L'encodeur par plan de bits encode successivement les plans de bits des magnitudes des coefficients (DC et AC) du plus significatif au moins significatif.

Pour chaque segment, un en-tête est d'abord encodé. Cet en-tête est composé de quatre parties, seule la première étant obligatoirement présente. Cette première partie contient certains *flags*, indique les parties optionnelles de l'en-tête présentes, ou encore contient des informations variant entre les segments. La deuxième partie contient le

nombre maximum d'octets compris dans le segment compressé et une limite de "qualité". Cette limite de qualité contraint la quantité d'information liée aux coefficients d'ondelette à encoder. Elle est spécifiée par un index de plan de bits et un point d'arrêt dans ce plan de bit. La troisième partie inclue des options de codage, notamment le nombre de blocs contenus dans le segment. Enfin, la quatrième partie contient des paramètres de l'image et de compression, fixes pour l'image entière. A noter que certaines parties de l'en-tête peuvent être présentes au début de l'image plutôt que pour chaque segment, si les informations qu'elles contiennent sont fixes pour l'image entière.

Puis un codage initial des coefficients DC quantifiés est réalisé. Les profondeurs de bits des coefficients AC sont ensuite codées. Enfin, chaque plan de bits est encodé, en commençant par le plus significatif et jusqu'au moins significatif. Le codage d'un plan de bits est réalisé en cinq étapes, numérotées de 0 à 4. Chaque étape est réalisée pour tous les blocs du segment avant de passer à l'étape suivante. L'étape 0 est consacrée aux coefficients DC tandis que les étapes 1 à 4 permettent de coder les coefficients AC.

Le compromis entre qualité de reconstruction et taille des données compressées est contrôlé pour chaque segment par le paramètre indiquant le nombre maximum d'octets compris dans ce segment compressé et/ou la limite de "qualité" liée aux coefficients d'ondelette. L'encodage de chaque segment est alors stoppé dès que la limite de taille du segment ou la limite de "qualité" est atteinte, selon celle se produisant la première.

Le *bitstream* de chaque segment est organisé de façon à obtenir une transmission progressive et peut ainsi être davantage tronqué en tout point afin de réduire le débit, mais au pris d'une parte de qualité du segment.

## 2.2 Codeurs vidéos en mode Intra

Après avoir décrit trois standards exclusivement destinés à compresser des images fixes, nous allons maintenant nous intéresser à deux standards de compression vidéo, utilisés en mode Intra. Ce mode, utilisé pour la compression des images de type $I$ au sein de vidéos, permet de compresser ces images indépendamment des autres images de la séquence. En mode Intra, un codeur vidéo appliqué à une image est ainsi équivalent à un codeur d'image fixe. Les standards auxquels nous allons nous intéresser sont H.264/AVC et son successeur HEVC.

### 2.2.1 H.264/AVC Intra

H.264/AVC [WSBL03] est un standard de compression de vidéos développé par le *ITU-T Video Experts Group* et le *ISO/IEC Moving Experts Group*, dont le draft final date de 2003.

Nous nous concentrerons ici sur la description de la couche de codage vidéo (VCL pour *Video Coding Layer*) et n'aborderons donc pas la couche d'abstraction au réseau (NAL pour *Network Abstraction Layer*) permettant l'usage de la VCL pour de nombreux systèmes.

Afin de tirer profit du fait que l'œil humain est plus sensible aux détails de luminosité que de couleur, H.264/AVC utilise un espace de couleur YCbCr pour lequel

la composante Y de *luma*, représentant la luminosité, est séparée des composantes de *chroma* Cb et Cr, représentant le niveau de déviation des couleurs du gris vers le bleu et le rouge respectivement. De plus, les composantes de chroma sont sous-échantillonnées par un facteur 2 sur les lignes et 2 sur les colonnes de sorte que chaque composante chroma ait quatre fois moins de valeurs que la composante de luma (format 4 :2 :0).

Une image est découpée en macroblocs de taille fixe 16x16 pour la composante de luma et 8x8 pour les composantes de chroma. Un macrobloc constitue l'unité de base sur laquelle le processus d'encodage et de décodage est appliqué. Les macroblocs sont regroupés en slices. Une image peut être découpée en une ou plusieurs slices, les slices se suffisant à elles-mêmes dans le sens où les informations qu'elles contiennent peuvent être décodées sans l'aide de données d'autres slices de l'image. Ainsi, il n'y a pas de prédiction Intra entre les slices.

Le processus d'encodage est appliqué pour chaque macrobloc (de luma ou de chroma) (Fig. 2.8). Les valeurs sont d'abord prédites grâce à des prédictions uniquement spatiales en Intra. Puis les résidus de prédiction sont transformés en utilisant une transformée entière sur des blocs 4x4. Les coefficients de la transformée sont ensuite quantifiés puis encodés par des méthodes de codage entropique.

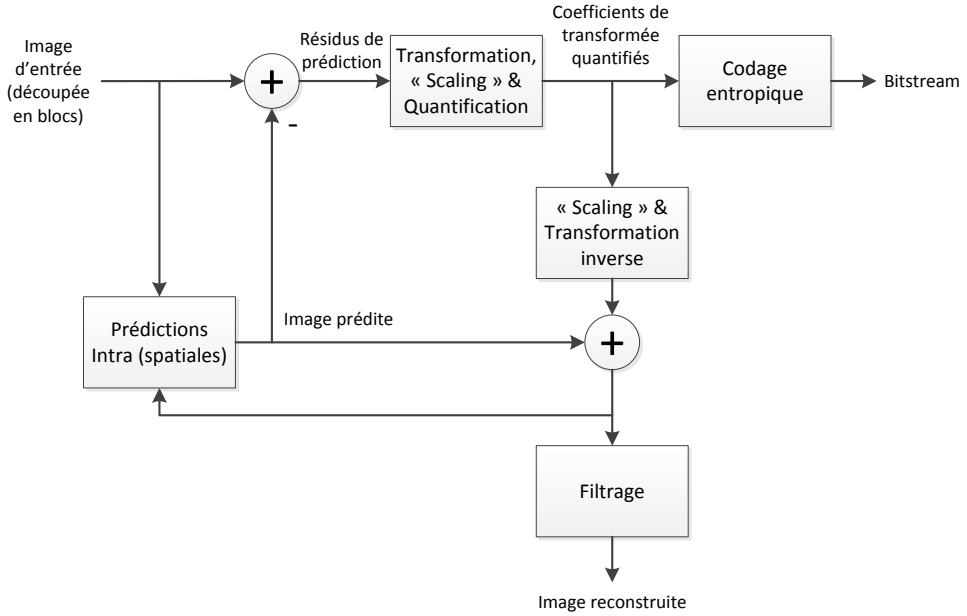



FIGURE 2.8 – Schéma simplifié d'un encodeur H.264/AVC en mode Intra.

#### 2.2.1.1 Les prédictions spatiales

Les prédictions spatiales permettent de prédire un bloc (4x4, 8x8 ou 16x16) à partir des valeurs voisines déjà décodées dans les blocs adjacents. Les prédictions sont toujours

réalisées dans le domaine spatial grâce aux valeurs voisines présentes à gauche ou au dessus du bloc à prédire. Plusieurs modes de prédiction existent, nommés Intra_4x4, Intra_16x16 et I_PCM. Le mode Intra_4x4 permet de prédire chaque bloc 4x4 de luma individuellement et est adapté aux zones de l'image présentant des détails. Le mode Intra_16x16 permet lui de prédire le macrobloc de luma entier et est plus adapté aux zones homogènes de l'image. Le troisième mode, I_PCM, permet d'éviter les étapes de prédiction, de transformée, de quantification et de codage entropique, les valeurs du bloc étant alors directement encodées. Cela permet à l'encodeur de représenter de manière précise les valeurs du bloc.

Avec le mode Intra_4x4, les 16 pixels du bloc à prédire sont prédits en utilisant les pixels A-Q déjà décodés appartenant à des blocs adjacents (Fig. 2.9). Un mode de prédiction est choisi pour chaque bloc 4x4, parmi neuf modes de prédiction existant, correspondant à huit modes de prédictions directionnels, particulièrement efficaces pour prédire des structures directionnelles au sein du bloc, et un mode "DC". Le mode 0, correspondant à une prédiction verticale, réalise une copie des valeurs A à D sur les colonnes correspondantes du bloc. Pour le mode 1, la prédiction horizontale, les valeurs des pixels I à L sont copiées sur les lignes correspondantes du bloc. Le mode DC réalise une moyenne des valeurs A à D et I à L et copie cette valeur moyenne dans tout le bloc. Les six autres modes correspondent à des prédictions diagonales et selon la direction, les pixels voisins appropriés sont sélectionnés et des combinaisons linéaires de leurs valeurs permettent de prédire les différents pixels du bloc. Ces prédictions peuvent être appliquées de façon similaire à un bloc 8x8.

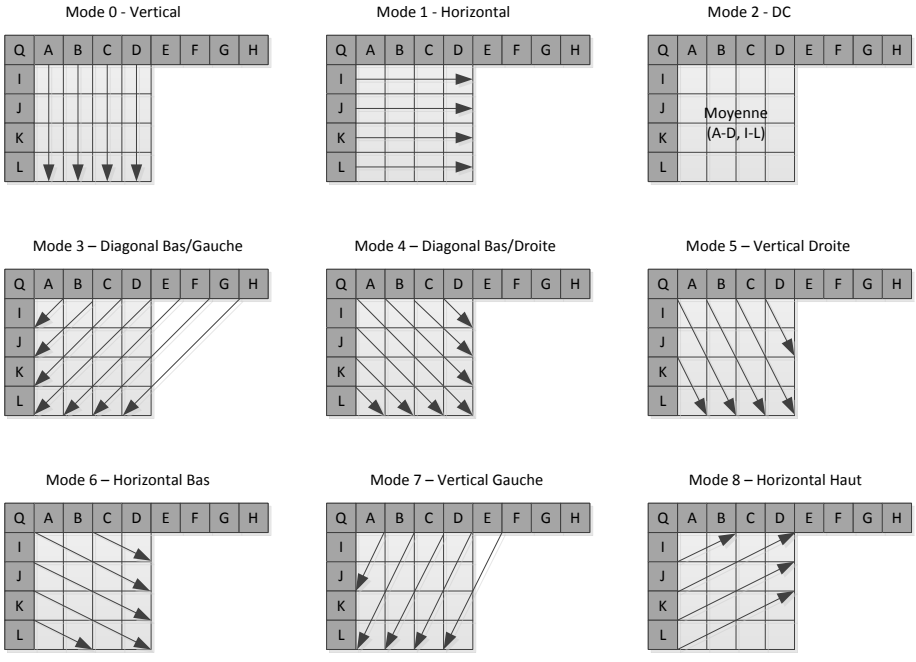



FIGURE 2.9 – Modes de prédictions Intra (spatiales) de H.264/AVC pour un bloc 4x4.

Avec le mode Intra_16x16, le macrobloc de luma complet est prédit. Il existe dans ce cas quatre modes de prédiction. Le mode 0 de prédiction verticale, le mode 1 de prédiction horizontale et le mode 2 de prédiction "DC" sont définis de la même façon que pour le mode Intra_4x4, mais avec davantage de voisins. Le mode 4, dit *plane*, construit quant à lui un plan linéaire à partir des pixels voisins de référence afin de prédire les valeurs du bloc 16x16.

Pour les valeurs de chroma d'un macrobloc, des prédictions similaires au cas Intra_16x16 sont réalisées.

Les prédictions temporelles ne sont pas décrites ici, étant donné qu'elles ne sont pas utilisées en mode Intra.

#### 2.2.1.2 La transformée entière

Les résidus de prédictions, correspondant à la différence entre le bloc original et sa prédiction, sont par la suite transformés puis codés. Dans H.264/AVC, la transformation est appliquée sur des blocs 4x4. Une transformée entière et séparable dont les propriétés sont similaires à celle d'une DCT 4x4 est utilisée. La matrice de transformation est la suivante :

$$H = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{pmatrix} \tag{2.5}$$

Le fait d'utiliser une transformée entière permet de ne pas perdre d'information en appliquant une transformée inverse au décodeur.

A noter que pour le mode de prédiction Intra_16x16 et pour les blocs de chroma, les coefficients DC subissent une seconde transformation.

#### 2.2.1.3 La quantification

Un paramètre de quantification (le QP) est utilisé au sein du standard pour définir la quantification appliquée aux coefficients issus de l'étape de transformation. Ce paramètre peut prendre 52 valeurs. Il permet de modifier le pas de quantification de telle sorte que l'augmenter de 1 permet d'augmenter le pas d'environ 12%. Autrement dit, ajouter 6 à ce paramètre permet de multiplier le pas de quantification par 2.

Les coefficients quantifiés d'un bloc sont ensuite scannés en "zig-zag", afin typiquement de regrouper dans un premier temps les forts coefficients de basses fréquences, puis les coefficients de hautes fréquences dont beaucoup sont nuls. Ces coefficients sont alors encodés par des méthodes de codage entropique.

#### 2.2.1.4 Le codage entropique

Afin d'encoder les coefficients de la transformée après quantification, deux méthodes de codage entropique peuvent être appliquées.

La première méthode est appelée *Context-Adaptive Variable Length Coding* (CAVLC) [WSBL03]. Elle encode le nombre de coefficients différents de zéros, ainsi que la taille

et la position de ces coefficients séparément, à l'aide de plusieurs tables comprenant des codes à longueur variable.

La seconde méthode, le *Context-Adaptive Binary Arithmetic Coding* (CABAC) [MSW03], permet d'améliorer encore davantage l'efficacité du codage entropique. L'utilisation du codage arithmétique permet d'assigner un nombre non entier de bits à chaque symbole. De plus, sa propriété de modélisation du contexte est importante et utilise les statistiques des éléments précédemment codés.

### 2.2.2 HEVC Intra

*High Efficiency Video Coding* (HEVC) [SOHW12, KMS+12, LBH+12] est le nouveau standard de compression vidéo du ITU-T *Video Coding Experts Group* et du ISO/IEC *Moving Picture Experts Group*, dont le draft final date de 2013. Successeur de H.264/AVC, son objectif est d'en améliorer les performances de compression, afin d'atteindre approximativement 50% de réduction de débit pour une perception visuelle de la vidéo de la même qualité. Le codage d'une vidéo (par la VCL) par le standard HEVC est réalisé par bloc avec une approche hybride, basée prédictions (spatiales et temporelles) et transformation 2D, comme H.264/AVC avant lui. Nous allons nous concentrer sur le mode Intra de HEVC dans le cas du codage d'une image fixe.

HEVC utilise typiquement l'espace de couleur YCbCr en 4 :2 :0, même si d'autres formats sont possibles.

Le processus d'encodage de HEVC Intra, réalisé par bloc, fonctionne sur les mêmes étapes principales que H.264/AVC Intra (Fig. 2.8), avec cependant un partitionnement des blocs plus complexe et plus efficace. Les blocs sont d'abord prédits via des prédictions spatiales. Puis, les résidus de prédiction, correspondant à la différence entre le bloc original et sa prédiction, sont ensuite transformés par une transformation linéaire. Les coefficients de la transformée sont ensuite "scalés", quantifiés et codés de façon entropique. Ils peuvent ainsi être transmis, accompagnés des informations de prédictions.

#### 2.2.2.1 Le partitionnement en quadtree

Dans H.264/AVC, l'image est découpée en macroblocs de taille fixe 16x16 pour la luma et 8x8 pour la chroma. Au sein de HEVC, l'image est découpée en *Coding Tree Units* (CTU). Une CTU consiste en un *Coding Tree Block* (CTB) de luma, des CTB de chroma et des éléments de syntaxe. Chaque CTB de luma peut avoir une taille variable de 16x16, 32x32 ou 64x64 pixels. Les CTB de chroma correspondants sont alors de tailles 8x8, 16x16 et 32x32 pixels respectivement. Le fait de pouvoir travailler avec des blocs de grande taille (jusqu'à 64x64) est particulièrement utile dans le cas de vidéos haute résolution.

Chaque CTU peut ensuite être partitionnée sur un modèle de quadtree, en une ou de multiples *Coding Units* (CU), la CTU étant la racine du quadtree. Une CU est composée d'un *Coding Block* (CB) de luma, des CB de chroma associés et d'éléments de syntaxe. Le découpage en quadtree s'effectue de façon itérative, chaque CB pouvant à nouveau

être découpé en quatre CB plus petits, et peut aller jusqu'à une taille minimale de CB de 8x8. Le découpage en quadtree est adapté au contenu de l'image : des zones homogènes favorisent l'utilisation de blocs de grande taille tandis que des détails imposent une décomposition en des blocs de plus petite taille. Chaque CU est par la suite partitionnée en *Prediction Units* (PU), et en un quadtree de *Transform Units* (TU).

La PU contient les informations de prédiction. Elle est formée des *Prediction Blocks* (PB) de luma et de chroma, ainsi que d'éléments de syntaxe relatifs à la prédiction. En Intra, la taille d'un PB est la même que celle du CB, excepté pour les CB ayant la taille minimale autorisée. Dans ce cas, le CB peut être partitionné en quatre PB dont la taille peut aller jusqu'à 4x4. La taille d'un *Prediction Block* (PB) peut donc varier de 4x4 pixels jusqu'à la taille du CB, pouvant atteindre 64x64 pixels.

Une CU est également décomposée de façon récursive en un quadtree de TU. Les feuilles de ce quadtree représentent les TU. L'étape de transformation des résidus de prédiction est réalisée sur les *Transform Blocks* (TB), des blocs carrés de tailles 4x4, 8x8, 16x16 ou 32x32.

La subdivision d'un CTB en CB et TB est illustrée sur la figure 2.10.

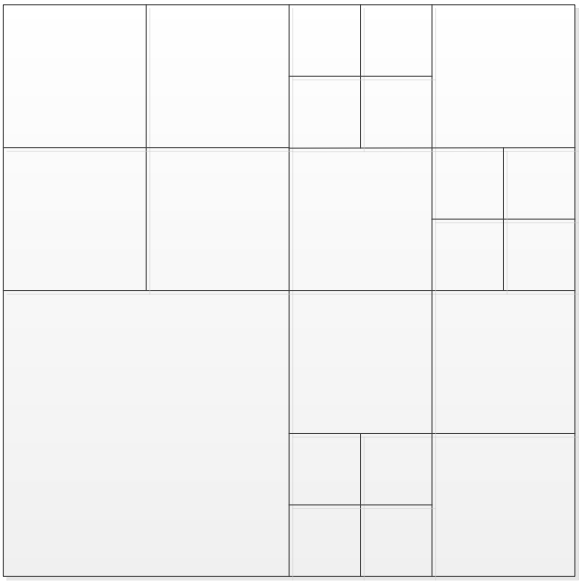


FIGURE 2.10 – Quadtree de subdivision d'un CTB en CB et TB.

On retrouve dans HEVC la notion de slice, déjà présente dans H.264/AVC. Une slice est une séquence de CTU traitées ligne par ligne, de gauche à droite ("raster scan"). Une image est ainsi décomposée en une ou plusieurs slices. Les slices se suffisent à elle même dans le sens où les informations qu'elles contiennent peuvent être décodées sans l'aide de données d'autres slices de l'image. Ainsi, il n'y a pas de prédiction Intra entre les slices. Le but premier des slices est la resynchronisation, dans le cas où des données seraient perdues.

La notion de tuile (*tile* en anglais) est également définie dans le standard. Une tuile

correspond à un ensemble rectangulaire de CTU. Les tuiles peuvent être décodées de façon indépendante et permettent donc l'usage d'architectures à traitements parallèles pour les opérations d'encodage et de décodage.

#### 2.2.2.2 Les prédictions spatiales

Les prédictions Intra réalisées dans le domaine spatial, déjà utilisées dans H.264/AVC, ont été étendues au sein de HEVC. Elles sont plus nombreuses et applicables à différentes tailles de blocs. Bien que le mode de prédiction spatial soit établi au niveau des PB, le processus de prédiction est réalisé pour chaque TB, selon les valeurs adjacentes déjà décodées appartenant aux TB voisins. On rappelle que la taille des TB peut aller de 4x4 à 32x32 valeurs. Il existe dans le standard 35 modes de prédictions différents (Fig. 2.11) : 33 modes directionnels correspondant à 33 orientations, ainsi que le mode DC et le mode planar. Les 33 modes directionnels permettent de prédire des régions présentant des contours directionnels importants. Là où H.264/AVC possède 8 modes de prédictions directionnels couvrant 8 directions, HEVC couvre 33 directions, ces directions étant davantage resserrées autour de la direction horizontale et de la direction verticale. Les valeurs reconstruites voisines du bloc NxN à prédire, servant de référence pour la prédiction, sont au nombre de $4N + 1$ et positionnées à gauche, au dessus, au dessus à droite et à gauche en bas (si elles sont disponibles) du bloc courant. Ces modes de prédiction directionnels sont applicables quelque soit la taille du TB. Le mode DC calcule une moyenne des valeurs voisines de référence pour la prédiction, tandis que le mode planar utilise les valeurs moyennes de deux prédictions linéaires utilisant quatre valeurs de référence.

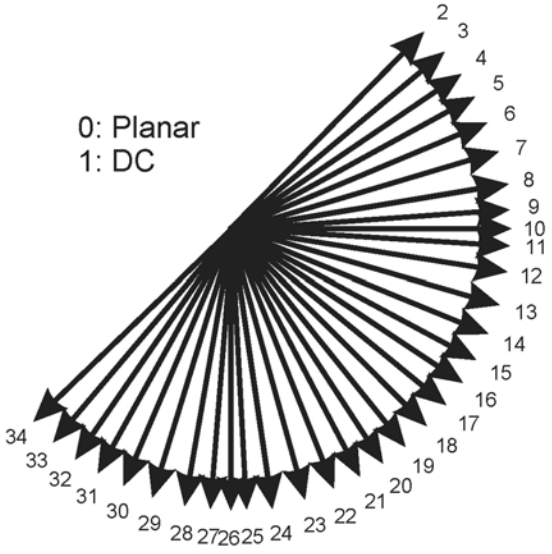



FIGURE 2.11 – Modes et directions des prédictions Intra (spatiales) de HEVC [SOHW12].

Pour les prédictions Intra des composantes de chroma, cinq modes peuvent être utilisés : deux modes directionnels (horizontal et vertical), le mode planar, le mode DC et un dernier mode permettant d'utiliser le même mode de prédiction directionnel que la composante de luma.

#### 2.2.2.3 La transformée

L'étape suivante est la transformation des résidus de prédiction. Cette étape est appliquée aux TB, de tailles 4x4, 8x8, 16x16 ou 32x32. La transformée 2D est appliquée via une transformée 1D horizontale puis verticale. Pour cela, une matrice dont les éléments sont dérivés des fonctions de base de la DCT est utilisée, après application d'un facteur multiplicatif et une approximation afin de ne contenir que des valeurs entières. Une seule matrice 32x32 est spécifiée, pour la transformation des blocs 32x32. Les matrices 4x4, 8x8 et 16x16 nécessaires à la transformation des autres blocs peuvent être dérivées de cette matrice 32x32 en y sélectionnant les valeurs adéquates. Une transformée alternative est appliquée aux blocs (TB) 4x4 de luma pour les modes de prédiction Intra. Cette transformation entière est dérivée d'une DST (Discrete Sinus Transform).

#### 2.2.2.4 La quantification

La quantification au sein de HEVC est, de façon similaire à celle de H.264/AVC, une quantification à reconstruction uniforme (URQ pour *Uniform Reconstruction Quantization*) contrôlée par un paramètre de quantification (le QP). Ce paramètre varie de 0 à 51 et une augmentation de 6 du QP correspond à une multiplication par 2 du pas de quantification.

#### 2.2.2.5 Le codage entropique

Contrairement à H.264/AVC, une seule méthode de codage entropique est définie dans HEVC : le CABAC [SB12].

Un aspect important du CABAC est sa modélisation du contexte. HEVC exploite, en plus des informations de contexte des voisins, les informations du quadtree, afin d'en déduire les indices des modèles de contexte des différents éléments de syntaxe.

HEVC adopte aussi une méthode de parcours des coefficients adaptative. Quelque soit la taille des TB, ce parcours est réalisé par sous-bloc de taille 4x4, au sein du TB. Pour les TB de tailles 4x4 et 8x8 en modes de prédiction Intra, trois méthodes de parcours des coefficients de transformée sont utilisées : diagonal, horizontal et vertical, selon la direction de la prédiction Intra. Pour les TB de tailles 16x16 et 32x32 en modes de prédiction Intra (ou pour toutes les tailles en modes de prédiction Inter), seule la méthode de parcours diagonal est appliquée.

Le cœur de la méthode de codage des coefficients reste inchangé par rapport à H.264/AVC. Le dernier coefficient non nul, une carte de signification, des bits de signe et les niveaux des coefficients de la transformée sont ainsi également transmis, mais divers changements sont appliqués aux différentes étapes. La position (via les coordonnées

horizontale et verticale) du dernier coefficient non nul du TB est d'abord transmise. Puis, la carte de signification permet d'indiquer les coefficients non nuls. Chaque sous-bloc 4x4 correspond à un groupe de signification. Pour tous les groupes comportant au moins un coefficient non nul précédant le dernier coefficient non nul du TB, un flag de signification du groupe indiquant un groupe non nul est transmis, suivi par un flag de signification pour chaque coefficient. Les signes des coefficients non nuls sont également transmis. Pour transmettre leur niveau, deux flags indiquant si ce niveau est plus grand que 1 ou 2 sont transmis, puis la valeur de niveau restante est codée selon la valeur des deux flags précédents.

## 2.3 Conclusion

Dans ce chapitre, nous avons présenté différents standards de compression, d'images fixes exclusivement ou de vidéos.

JPEG, grâce à son algorithme relativement simple, est un codec de compression d'images devenu très populaire. JPEG 2000 permet d'obtenir de meilleures performances de compression par rapport à JPEG, mais a eu du mal à se démocratiser, notamment du fait de son codage entropique plus complexe. CCSDS 122 est lui spécialisé pour la compression d'images spatiales. Proche de JPEG 2000 dans son fonctionnement, des compromis ont toutefois été réalisés afin de maîtriser la complexité.

Deux standards de compression de vidéos ont également été présentés. H.264/AVC est un standard très répandu aujourd'hui. HEVC, récemment standardisé, se positionne comme son successeur et en améliore les performances. Des extensions à HEVC vont prochainement voir le jour, afin notamment de traiter les cas du codage scalable, de formats à haute dynamique ou encore du multi-vues/3D.

Ces différents standards de compression vont nous servir de références lors d'expérimentations dans la deuxième partie de cette thèse.

# Deuxième partie

# Apprentissage de dictionnaires structurés pour la compression d'images satellites

# Chapitre 3

# D'une structure arborescente à une structure adaptative

Après avoir présenté dans le première partie de cette thèse les concepts de représentations parcimonieuses et d'apprentissage de dictionnaire, avec un état de l'art des différentes méthodes et applications, puis décrit plusieurs standards de compression, nous allons présenter nos contributions dans les deux autres parties de cette thèse.

Dans cette deuxième partie, nous nous intéresserons à l'apprentissage de dictionnaires structurés dans le cadre de la compression d'images satellites. Pour cela nous allons dans ce chapitre étudier différentes structures de dictionnaires adaptées au codage, puis nous les testerons dans le cadre de la compression dans le chapitre suivant. Enfin, dans le dernier chapitre de cette deuxième partie, nous chercherons à apprendre des dictionnaires spécialisés de façon à améliorer leurs performances de représentation.

## 3.1 Description du problème

Le problème auquel nous sommes confrontés est la compression d'images spécifiques : les images satellites. Ces images sont captées par des satellites et nécessitent d'être compressées à bord avant d'être envoyées sur Terre, où elles sont décompressées puis traitées avant d'être utilisées. Afin de coder ces images, plutôt que d'utiliser des transformations prédéfinies comme les codeurs standards, une transformée spécifique est utilisée par le biais de l'apprentissage de dictionnaires pour les représentations parcimonieuses. L'idée est d'adapter les dictionnaires, et donc la transformée, à ce type d'images dans le but d'en améliorer la représentation. Utilisées dans le cadre de la compression, les représentations parcimonieuses nécessitent de coder et transmettre pour chaque bloc de l'image à compresser un certain nombre de paires coefficient-indice, l'indice indiquant l'atome du dictionnaire utilisé dans la représentation.

Un exemple commun d'algorithme d'apprentissage, que nous avons évoqué dans le premier chapitre, est K-SVD [AEB06]. Mais ce type d'algorithme présente deux limitations majeures. Premièrement, le dictionnaire est appris pour une valeur de parcimonie spécifique, et n'est donc pas optimal dans le cas d'une utilisation pour une autre valeur

de parcimonie. On dira que le dictionnaire n'est pas scalable en parcimonie. A noter que l'algorithme peut également apprendre le dictionnaire pour une certaine erreur de représentation admise pour représenter chaque signal d'apprentissage. Deuxièmement, le coût de codage de chaque indice d'atome sélectionné lors de la décomposition est directement impacté par la taille du dictionnaire. En effet, augmenter le nombre d'atomes du dictionnaire afin d'améliorer sa capacité de représentation entraîne également une hausse du coût de codage de chaque indice. Ce type de dictionnaire, que l'on appellera "plat", n'est donc pas forcément optimal pour une réduction du coût codage.

C'est pourquoi l'objectif est d'apprendre une structure de dictionnaires (ou un dictionnaire structuré) davantage adaptée au problème du codage des indices, et scalable en parcimonie, offrant ainsi un bon compromis entre erreur de représentation et coût de codage. Cette structuration du dictionnaire doit nous permettre de pouvoir augmenter le nombre global d'atomes qu'il contient afin d'améliorer la qualité des représentations, tout en conservant un coût de codage des indices réduit, car nous sommes dans le cadre de la compression, et une complexité de la décomposition, celle-ci étant réalisée à bord du satellite, équivalente.

D'après cet objectif et dans la lignée des travaux sur les ITD [ZGK+10, Zep10], nous allons dans ce chapitre étudier différentes structures de dictionnaires, où, d'après le concept des ITD, chaque niveau est utilisé pour une itération de l'algorithme de décomposition. Nous présenterons tout d'abord une structure arborescente nommée "Tree K-SVD", similaire à la structure TSITD. Puis nous ferons évoluer cette structure vers une structure en "cerf-volant", correspondant à une structure en arbre sur les premiers niveaux, mais dont les branches sont ensuite refermées afin de n'apprendre qu'un seul dictionnaire par niveau. Enfin, nous étudierons une structure adaptative, dont la structure s'adapte automatiquement durant l'apprentissage aux données d'entraînement, en refermant progressivement les différentes branches en une branche commune. Nous montrerons ensuite l'intérêt du retrait de la moyenne des patchs d'apprentissage et de test. Puis nous clôturerons ce chapitre en abordant le critère d'arrêt de la décomposition en testant une erreur cible plutôt qu'une parcimonie cible.

## 3.2 Tree K-SVD : une structure arborescente

La première structure à laquelle nous allons nous intéresser est une structure arborescente nommée "Tree K-SVD" [AMGL13a], similaire à la structure du TSITD [ZGK11, Zep10].

Cette structure est apprise sur $L$ niveaux. Elle compte un unique dictionnaire de $K$ atomes au premier niveau. Chaque atome à un niveau donné possède un dictionnaire fils, il y a donc $K$ dictionnaires au deuxième niveau, $K^2$ au troisième, ... et $K^{i-1}$ au niveau $i$ (Fig. 3.1).

### 3.2.1 Apprentissage de la structure

L'apprentissage de la structure est réalisé niveau par niveau, de haut en bas, dictionnaire par dictionnaire. Chaque dictionnaire de l'arbre est appris sur des résidus du

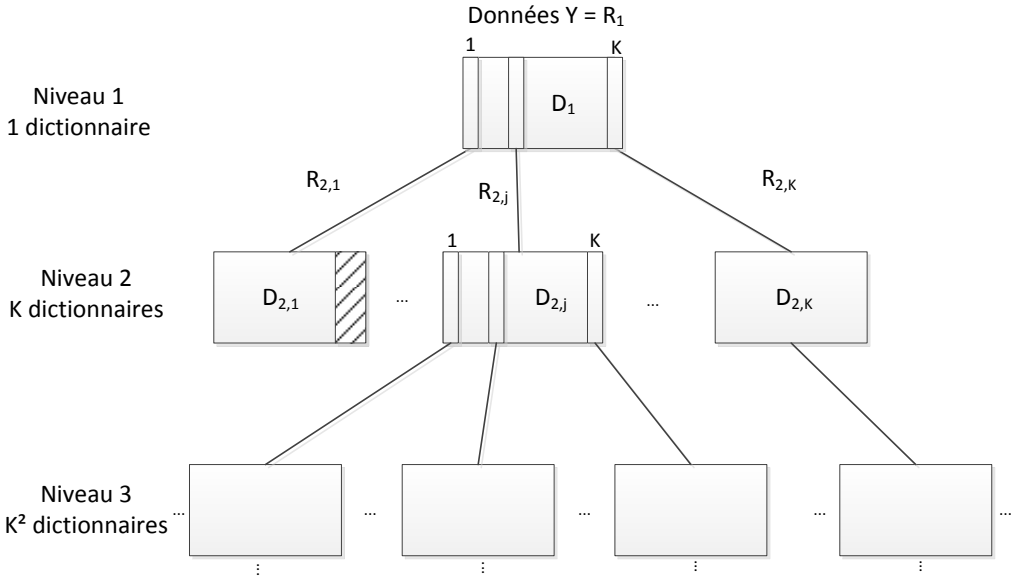



FIGURE 3.1 – La structure arborescente Tree K-SVD.

niveau supérieur par l'algorithme K-SVD [AEB06], avec une contrainte de parcimonie fixée à 1 atome. Cette contrainte de parcimonie à l'apprentissage est fixée à 1 atome étant donné que lors de la décomposition, 1 atome sera sélectionné par dictionnaire utilisé. Le fait d'utiliser K-SVD avec une parcimonie de 1 atome implique que la SVD, appliquée normalement à une matrice d'erreur $E_k^R$ pour la mise à jour de chaque atome $d_k$ et des coefficients associés, est ici directement appliquée aux données d'apprentissage du dictionnaire, restreintes à celles utilisant l'atome $d_k$ dans leur décomposition. A noter que l'apprentissage de chaque dictionnaire au sein de la structure pourrait très bien être réalisé par une autre méthode d'apprentissage de dictionnaire.

Au premier niveau, l'unique dictionnaire $D_1$ est appris sur l'ensemble des données d'apprentissage $Y$ (correspondant à l'ensemble des résidus $R_1$), composé de $N$ vecteurs dans $\mathbb{R}^n$, avec K-SVD pour une parcimonie de 1 atome. Puis, l'algorithme MP [MZ93] est appliqué avec une parcimonie de 1 atome également afin d'approximer chaque vecteur de $Y$ par seulement 1 atome de $D_1$ (avec un coefficient associé). Suite à cette approximation, l'ensemble des résidus $R_2$ est calculé :

$$R_2 = Y - D_1 X \; avec \; \forall i = 1, ..., N, ||x_i||_0 = 1 \tag{3.1}$$

$x_i$ représentant la colonne $i$ de $X$, $X$ étant calculé à l'aide de l'algorithme MP appliqué pour une parcimonie de 1 atome.

Puis $R_2$ est partitionné en $K$ ensembles de résidus, $R_{2,1}, ..., R_{2,K}$, selon l'atome de $D_1$ choisi pour l'approximation au premier niveau. Ainsi, tout vecteur de $R_2$ est placé dans l'ensemble $R_{2,k}$, $k \in [1, ..., K]$, s'il est le résidu de l'approximation d'un vecteur de $Y$ par l'atome $k$ de $D_1$. Autrement dit, les résidus liés à l'atome $k$, $R_{2,k}$, correspondent aux résidus des vecteurs d'entrainement de $Y$ qui sont le plus corrélés

à l'atome $k$ du dictionnaire $D_1$. Au deuxième niveau, un dictionnaire est appris sur chaque ensemble de résidus $R_{2,k}$, afin d'apprendre $K$ dictionnaires. Chaque ensemble de résidus $R_{2,k}$ est alors approximé par un atome du dictionnaire correspondant $D_{2,k}$ avec l'algorithme MP. Les résidus sont mis à jour et partitionnés en $K$ ensembles de résidus pour chaque dictionnaire $D_{2,k}$ pour ainsi former $K^2$ ensembles de résidus au troisième niveau, utilisés pour apprendre les $K^2$ dictionnaires de ce troisième niveau. De cette façon, chaque dictionnaire à un niveau donné est adapté à un sous-ensemble de résidus du niveau précédent. L'apprentissage est ainsi réalisé jusqu'à atteindre le niveau souhaité.

Certains dictionnaires de l'arbre peuvent être vides dans le cas où l'atome père au niveau supérieur, appartenant au dictionnaire père, n'est utilisé dans aucune approximation des résidus du niveau supérieur liés au dictionnaire père. Certains dictionnaires peuvent également être incomplets, c'est-à-dire comporter moins de $K$ atomes (comme c'est le cas de $D_{2,1}$ sur la figure 3.1), si le nombre de vecteurs de résidus permettant d'apprendre ces dictionnaires est strictement inférieur à $K$. Dans ce cas, les vecteurs de résidus sont simplement copiés dans le dictionnaire, avec une normalisation afin que les atomes du dictionnaires aient bien une norme $l_2$ unitaire. L'apprentissage de la branche est alors arrêté, c'est-à-dire que tout dictionnaire incomplet n'a pas de dictionnaire fils.

### 3.2.2 Décomposition au sein de la structure

Une fois qu'un dictionnaire en arbre Tree K-SVD de $L$ niveaux a été appris sur des données d'entraînement, il peut être utilisé pour approximer tout vecteur test par une combinaison linéaire de $l$ atomes du dictionnaire, avec $l \leq L$, 1 atome étant sélectionné par niveau, de haut en bas. La contrainte de parcimonie $l$ indique donc jusqu'à quelle profondeur de l'arbre la sélection des atomes est effectuée. Le choix d'un atome au sein d'un dictionnaire, et le calcul du coefficient associé, est réalisé grâce à l'algorithme MP, comme à l'apprentissage. L'algorithme MP est donc appliqué à chaque niveau sur un dictionnaire et le vecteur de résidus courant, pour une parcimonie de 1 atome. L'atome sélectionné à un niveau indique dans quel dictionnaire chercher l'atome suivant au niveau inférieur, chaque atome à un niveau donné possédant un unique dictionnaire fils au niveau suivant.

Prenons l'exemple d'un vecteur test $y$ que l'on cherche à approximer par trois atomes du dictionnaire (Fig. 3.2). L'algorithme MP est d'abord appliqué sur $y$ (correspondant aux résidus $r_1$) et $D_1$ avec une parcimonie de 1 atome afin d'approximer $y$ par un atome de $D_1$ au premier niveau et de calculer le coefficient associé. L'algorithme MP fournit ainsi l'atome de $D_1$ le plus corrélé à $y$, que l'on nomme $d_1$, et le coefficient associé $x_1$. Les résidus sont alors calculés :

$$r_2 = y - d_1 x_1 \tag{3.2}$$

L'atome $d_1$ ayant été sélectionné au premier niveau, la recherche du deuxième atome est effectuée au sein du dictionnaire fils de $d_1$ : $D_2$. L'algorithme MP est de nouveau appliqué, cette fois sur le vecteur de résidus $r_2$ et le dictionnaire $D_2$ pour une parcimonie de 1 atome, afin de sélectionner l'atome de $D_2$ le plus corrélé à $r_2$, $d_2$, et de calculer le

coefficient associé $x_2$. Les résidus sont alors mis à jour :

$$r_3 = r_2 - d_2 x_2 \tag{3.3}$$

Le choix de $d_2$ au deuxième niveau indique de chercher le troisième atome au sein de $D_3$, le dictionnaire fils de $d_2$. L'algorithme MP permet de trouver $d_3$ et de calculer $x_3$ et les résidus sont mis à jour une dernière fois :

$$r = r_3 - d_3 x_3 \tag{3.4}$$

Finalement, le vecteur test $y$ peut être écrit comme une approximation de la combinaison linéaire des trois atomes $d_1$, $d_2$ et $d_3$ :

$$y \approx d_1 x_1 + d_2 x_2 + d_3 x_3 \tag{3.5}$$

et $r$ correspond aux résidus finaux de la représentation de $y$ par cette combinaison linéaire.

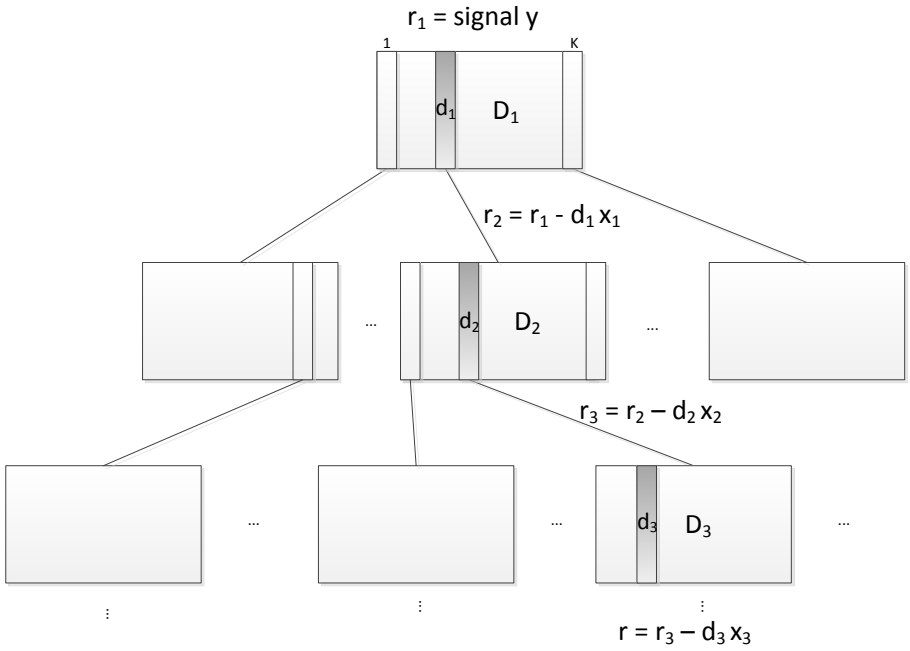



FIGURE 3.2 – Décomposition sur la structure Tree K-SVD.

### 3.2.3 Intérêts de la structure

Cette structure arborescente, bien que comptant un nombre global d'atomes important, est composée de multiples petits dictionnaires. Elle permet ainsi un codage efficace des indices étant donné que ce sont les indices parmi de petits dictionnaires qui

sont codés et qu'il n'est pas nécessaire de coder l'information du dictionnaire à utiliser à chaque niveau, l'indice de l'atome sélectionné à un niveau indiquant quel dictionnaire utiliser au niveau suivant. Cette structuration permet également une recherche rapide des atomes dans la structure lors de la décomposition, chaque atome étant sélectionné parmi un unique dictionnaire de la structure. Cette structure arborescente, composée de multiples dictionnaires de $K$ atomes, permet donc d'augmenter le nombre global d'atomes qu'elle contient pour améliorer sa capacité de représentation, tout en conservant un coût de codage de chaque indice codé individuellement et une complexité de décomposition équivalents à un dictionnaire "plat" de $K$ atomes, pour lequel les $l$ atomes sont sélectionnés lors de la décomposition dans le même dictionnaire. Ces deux critères de coût de codage des indices et de complexité de décomposition sont importants dans le cadre d'une compression réalisée à bord du satellite. Dans ce cadre, nous considérerons le dictionnaire (structuré ou non) connu du codeur comme du décodeur (voir Chapitre 4).

Enfin, cette structure arborescente est scalable en parcimonie, c'est-à-dire adaptée à plusieurs valeurs de parcimonie. En effet, la structure est apprise pour un nombre de niveaux donné et peut ensuite être utilisée pour toute contrainte de parcimonie (limitant le nombre d'atomes sélectionnés dans la représentation) inférieure ou égale au nombre de niveaux. Cela peut être utile dans le cas où les différents vecteurs test sont représentés pour une parcimonie variable.

### 3.2.4 Expérimentations

Afin de montrer les performances des dictionnaires structurés vis-à-vis des dictionnaires "plats", nous allons dans ce chapitre comparer leur qualité de représentation en fonction de la parcimonie utilisée, c'est-à-dire en fonction du nombre d'atomes utilisés dans la représentation de chaque vecteur test.

Les dictionnaires sont d'abord appris sur une même base de treize images satellites, fournies par *Airbus Defence and Space*, représentant des scènes variées telles que des villes, de la mer, du désert, des champs... (Annexe 1). Des patchs de $8 \times 8$ pixels sont extraits de ces images afin de former un ensemble de 654688 vecteurs d'apprentissage. Les dictionnaires sont initialisés avec le dictionnaire DCT complet ou sur-complet. Les dictionnaires "plats" sont appris en 50 itérations. Pour les structures sur plusieurs niveaux, 50 itérations sont réalisées pour apprendre le dictionnaire au premier niveau, puis 10 pour les dictionnaires des niveaux suivants afin d'accélérer le processus d'apprentissage. Le premier niveau étant le plus important, plus d'itérations lui sont accordées, l'impact sur les résultats reste ainsi limité. Le dictionnaire K-SVD est appris en utilisant le code fourni par les auteurs [AEB].

Dans un contexte de compression, nous ne cherchons pas à comparer les dictionnaires avec le même nombre total d'atomes, mais à un coût de codage de chaque indice équivalent, ainsi qu'à complexité de décomposition comparable. Ainsi, les dictionnaires plats comportent $K$ atomes, tandis que pour les structures de dictionnaires, $K$ représente le nombre d'atomes de chaque dictionnaire au sein de la structure.

L'image test, *New York* (Fig. 3.3), est une image 8 bits en niveaux de gris de

$2400 \times 2400$ pixels, contenant à la fois des zones texturées et d'autres plus homogènes. L'image est découpée en patchs de $8 \times 8$ pixels ne se chevauchant pas, afin de former 90 000 vecteurs test. Chaque vecteur test est alors décomposé pour plusieurs valeurs de parcimonie sur les différents dictionnaires à comparer. Pour chaque valeur de parcimonie, le PSNR entre l'image reconstruite et l'image originale est ensuite calculé (en dB).

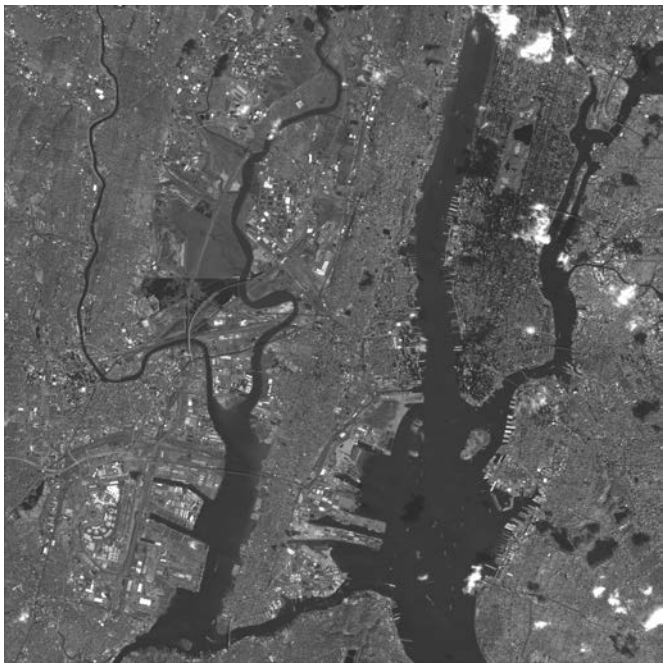


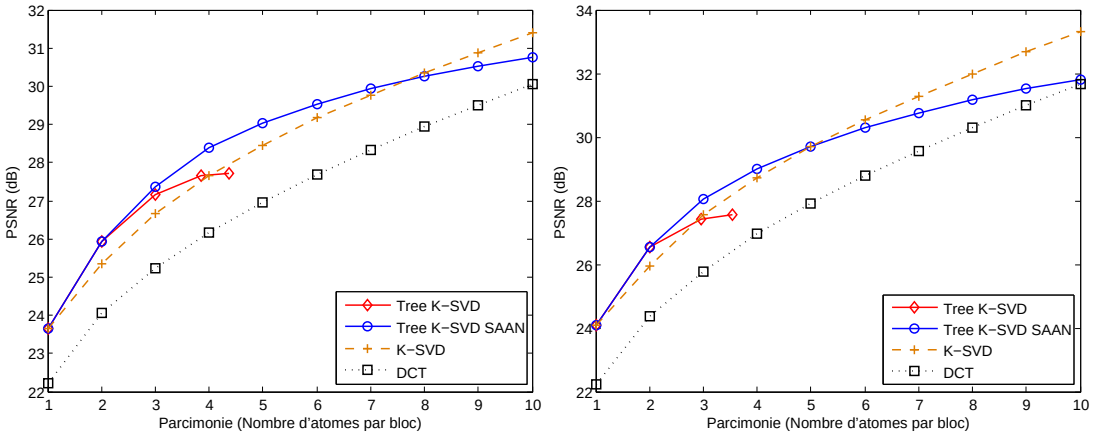
FIGURE 3.3 – Image test : *New York* (2400x2400) (FTM de 0.35).

Dans cette première comparaison, la structure en arbre Tree K-SVD est comparée au dictionnaire "plat" appris avec K-SVD de façon optimale, un dictionnaire K-SVD étant appris pour chaque valeur de parcimonie, afin que l'apprentissage et la décomposition puissent être effectués pour la même contrainte de parcimonie. Le dictionnaire DCT (transformée introduite dans [ANR74]) (DCT 2D de type 2) prédéfini est également ajouté à la comparaison afin d'évaluer l'apport de l'apprentissage. Il est utilisé comme un dictionnaire dans lequel on viendra sélectionner plusieurs atomes. L'algorithme OMP [PRK93] est utilisé pour la décomposition sur les dictionnaires K-SVD et DCT, pour chaque valeur de parcimonie. Cela signifie que la décomposition sur les dictionnaires "plats" est plus complexe, du fait du calcul de la pseudo-inverse au sein de l'algorithme OMP, que la décomposition sur les dictionnaires structurés où un simple algorithme MP est réalisé à chaque niveau. A noter que l'algorithme OMP sera également adapté aux dictionnaires structurés lors de tests intervenant plus tard dans ce chapitre. Les tests sont réalisés sur des dictionnaires complets (K=64) ou sur-complets (K=256).

La première observation (Fig. 3.4) est le gain en qualité de représentation du dictionnaire appris K-SVD par rapport au dictionnaire DCT prédéfini. Cela montre l'intérêt

d'apprendre un dictionnaire.

La structure en arbre Tree K-SVD offre de meilleures performances que le dictionnaire "plat" K-SVD lorsque la contrainte de parcimonie est relativement faible (Fig. 3.4), c'est-à-dire tant que la sélection des atomes au sein de l'arbre est effectuée sur les premiers niveaux de l'arbre. Pour une parcimonie de 1 atome, seul le premier niveau de l'arbre est utilisé, Tree K-SVD est alors équivalent à un dictionnaire "plat" appris avec K-SVD pour une parcimonie de 1 atome. Tree K-SVD et K-SVD donnent alors logiquement le même résultat. Puis, pour une parcimonie de 2 ou 3 atomes (pour K=64), Tree K-SVD offre de meilleurs résultats que K-SVD. Cependant, en augmentant par la suite le nombre d'atomes dans la représentation, le gain en PSNR devient faible et la courbe s'arrête rapidement, dépassée par celle de l'algorithme K-SVD.




(a) K = 64 (b) K = 256

FIGURE 3.4 – Tests PSNR-parcimonie de Tree K-SVD pour K=64 (a) et K=256 (b) sur *New York*.

### 3.2.5 Limitations

Bien que Tree K-SVD soit efficace sur les premiers niveaux, ses performances stagnent lorsque la contrainte de parcimonie augmente. En effet, le nombre de dictionnaires par niveau croît rapidement et devient très important après seulement quelques niveaux. Un problème d'*overfitting* apparaît alors, les niveaux profonds de l'arbre devenant trop adaptés aux données d'apprentissage. Le nombre de dictionnaires se multipliant à chaque niveau, les vecteurs de résidus sont éparpillés entre les trop nombreuses branches et beaucoup de dictionnaires se retrouvent incomplets, voire totalement vides, d'où une perte d'efficacité des niveaux profonds de l'arbre. De plus, l'apprentissage des branches s'arrêtant suite à un dictionnaire incomplet ou vide, les branches s'arrêtent après seulement quelques niveaux et il devient alors impossible d'y sélectionner davantage d'atomes, d'où l'arrêt prématuré de la courbe (Fig. 3.4).

### 3.2.6 La Sélection Adaptative des Atomes par Niveau (SAAN)

Afin de tirer profit des premiers niveaux de l'arbre, qui sont les niveaux les plus efficaces, une méthode de décomposition alternative sélectionnant les atomes de manière adaptative à chaque niveau est proposée . Ainsi, plutôt que de se restreindre à sélectionner 1 atome seulement par niveau, cette méthode permet de sélectionner 1 ou plusieurs atomes par niveau. Davantage d'atomes que le nombre de niveaux de l'arbre peuvent de cette manière être sélectionnés pour la représentation de chaque vecteur test.

A chaque niveau (Fig. 3.5 (a)), une fois que le premier atome ($d_1$) a été sélectionné, un choix est réalisé entre sélectionner un deuxième atome au sein du même dictionnaire en restant au niveau courant ou sélectionner le prochain atome au niveau suivant de l'arbre. Pour cela, les deux atomes les plus corrélés au vecteur de résidus courant $r$ sont trouvés, un au niveau courant dans le même dictionnaire $D_i$ ($c_1$ : choix d'atome n°1) et un dans le dictionnaire fils $D_{i+1}$ au niveau suivant ($c_2$ : choix d'atome n°2). L'atome minimisant l'énergie des résidus (calculés en ajoutant ce nouvel atome dans la représentation) est sélectionné dans la représentation :

$$d^* = \arg\min_c \left(||r_1||_2^2, ||r_2||_2^2\right) \tag{3.6}$$

avec :

$$r_1 = r - c_1 x_{c_1} \tag{3.7}$$

$$r_2 = r - c_2 x_{c_2} \tag{3.8}$$

Si le nouvel atome est sélectionné dans le même dictionnaire (l'atome $c_1$ est donc choisi pour devenir le deuxième atome $d_2$ sélectionné à ce niveau, Fig. 3.5 (b)), le choix du prochain atome est alors réalisé de la même manière entre un troisième atome du même dictionnaire $D_i$ ($c_1$ : nouveau choix d'atome n°1) ou un atome au niveau suivant au sein du dictionnaire fils $D_{i+1}$ ($c_2$ : nouveau choix d'atome n°2). Le choix est donc toujours réalisé entre rester au même niveau ou aller au niveau suivant, jusqu'à atteindre la parcimonie souhaitée. La parcimonie par niveau est ainsi automatiquement adaptée de façon à diminuer la distorsion. A noter que le dictionnaire fils $D_{i+1}$ où la décomposition sera poursuivie au niveau $i+1$ est déterminé par le premier atome sélectionné dans le dictionnaire courant $D_i$.

Étant donné que plusieurs atomes peuvent être sélectionnés par niveau, la SAAN permet de sélectionner davantage d'atomes au sein de la même structure apprise pour représenter chaque vecteur test, afin d'atteindre une meilleure qualité de représentation (Fig. 3.4). Tree K-SVD SAAN peut donc atteindre une parcimonie de 10 atomes, contrairement à Tree K-SVD. De plus, le fait d'avoir la possibilité de choisir un nouvel atome au sein du même dictionnaire ou au sein du dictionnaire fils permet d'augmenter les possibilités de représentation et améliore ainsi les performances par rapport à la décomposition classique sélectionnant 1 atome par niveau. Cette sélection adaptative permet aussi de sélectionner davantage d'atomes sur les premiers niveaux efficaces de l'arbre, mais la décomposition atteint tout de même les niveaux plus profonds et moins efficaces. Une fois la fin d'une branche atteinte, la sélection des atomes se retrouve bloquée au sein du dernier dictionnaire de cette branche et l'amélioration de qualité

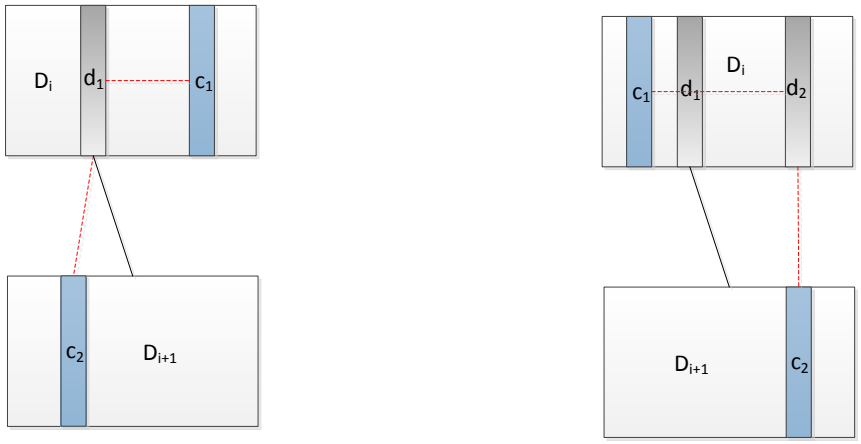



FIGURE 3.5 – Schéma de la Sélection Adaptative des Atomes par Niveau (SAAN) : (a) étape 1 (gauche) (b) étape 2 (droite).

due à l'ajout d'un nouvel atome est de moins en moins importante. Les performances en PSNR de K-SVD finissent donc par rejoindre et dépasser celles de Tree K-SVD SAAN lorsque davantage d'atomes sont sélectionnés.

Mais la SAAN implique de ne plus être à un coût de codage des indices équivalent puisqu'un *flag* de 1 bit par atome doit être ajouté au *bitstream* afin de savoir si le prochain atome est sélectionné au niveau courant ou au niveau suivant. De même, le fait de chercher le prochain atome parmi le dictionnaire courant et un dictionnaire au niveau suivant augmente la complexité de la décomposition, de sorte que la comparaison n'est plus effectuée à complexité comparable. C'est pourquoi nous allons désormais nous concentrer sur l'évolution de la structure en elle-même, tout en revenant à une décomposition sélectionnant 1 atome par niveau, en cherchant à obtenir de meilleures représentations mais sans surplus de coût de codage et de complexité.

## 3.3 La structure en "cerf-volant"

Afin d'éviter le fait d'avoir trop de dictionnaires dans les niveaux profonds de l'arbre, responsable d'une perte d'efficacité de la structure lorsque la contrainte de parcimonie augmente, une évolution de la structure Tree K-SVD, nommée structure en "cerf-volant" [AMGL13c], est présentée.

La structure en "cerf-volant" (Fig. 3.6) est identique à la structure Tree K-SVD sur les premiers niveaux, mais à partir d'un certain niveau, les différentes branches de l'arbre sont refermées pour n'apprendre plus qu'un seul dictionnaire par niveau. Tout comme Tree K-SVD, l'algorithme K-SVD est utilisé pour apprendre chaque dictionnaire de la structure, avec une parcimonie de 1 atome.

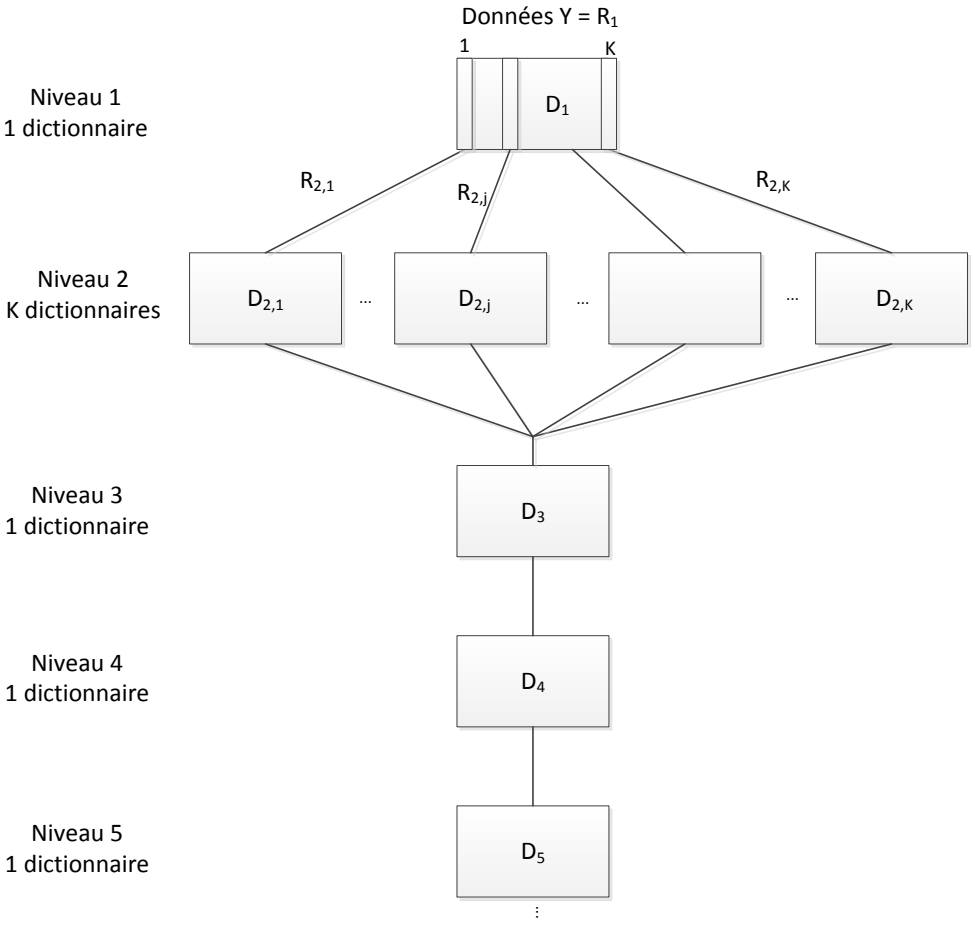



FIGURE 3.6 – La structure en "cerf-volant" (exemple de paramétrage imposant une fermeture après le deuxième niveau).

### 3.3.1 Apprentissage de la structure

La structure en "cerf-volant" est apprise comme une structure en arbre Tree K-SVD jusqu'à un niveau donné $i$. Les résidus issus de tous les dictionnaires complets au niveau $i$ sont alors fusionnés afin d'apprendre un unique dictionnaire sur l'ensemble de ces résidus au niveau suivant $i+1$. Ce dictionnaire constitue le premier dictionnaire de la "traîne du cerf-volant". Dès lors, un seul dictionnaire est appris par niveau, sur les résidus du dictionnaire au niveau précédent.

A noter que sur la première partie en arbre de la structure en "cerf-volant", certaines branches peuvent s'arrêter après un dictionnaire incomplet avant d'atteindre le niveau $i$. Dans ce cas là, les résidus du dictionnaire, quasiment nuls car le dictionnaire incomplet correspond à une copie avec normalisation des vecteurs de résidus, ne sont pas pris en compte pour l'apprentissage de la "traîne du cerf-volant", et le dernier dictionnaire de la branche arrêtée possède le premier dictionnaire de la "traîne", au niveau $i+1$, comme dictionnaire fils.

### 3.3.2 Décomposition au sein de la structure

Une fois la structure en "cerf-volant" apprise, elle peut être utilisée pour approximer un signal comme une combinaison linéaire d'atomes de cette structure. Comme pour Tree K-SVD, un atome est sélectionné par niveau et un coefficient associé est calculé, grâce à l'algorithme MP appliqué pour une parcimonie de 1 atome sur un dictionnaire à chaque niveau. La sélection des atomes est d'abord réalisée au sein de la partie en arbre, le choix d'un atome à un niveau donné indiquant le dictionnaire à utiliser au niveau suivant, puis se poursuit sur la "traîne", jusqu'à ce que la parcimonie souhaitée, ou le dernier niveau de la structure, soit atteint.

Dans le cas où une branche sur la partie en arbre s'arrête avant le niveau $i$, alors la décomposition se poursuit sur la "traîne du cerf-volant", à partir du niveau $i+1$. Dans ce cas précis, il est possible qu'un ou plusieurs niveaux, présentant un dictionnaire vide suite à l'arrêt de la branche, soient passés (c'est-à-dire qu'aucun atome n'est sélectionné à ces niveaux), pour continuer la décomposition sur la "traîne".

### 3.3.3 Intérêts par rapport à la structure arborescente

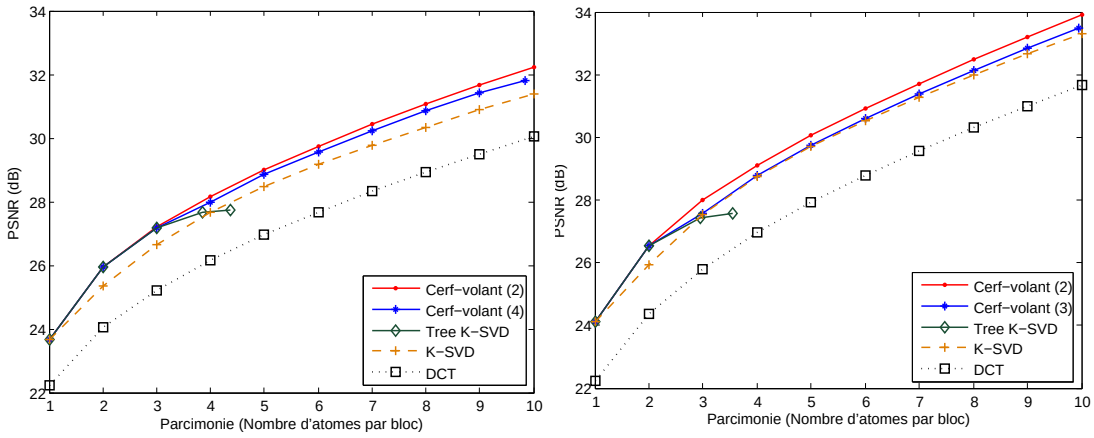
La structure en "cerf-volant" présente les mêmes avantages que la structure arborescente. Composée de multiples dictionnaires de $K$ atomes, elle peut globalement contenir un nombre important d'atomes pour un coût de codage de chaque indice ainsi qu'une complexité de décomposition équivalents par rapport à un dictionnaire "plat" de $K$ atomes. De plus, elle est également scalable en parcimonie, chaque dictionnaire étant toujours appris sur des résidus du niveau supérieur pour une parcimonie unitaire.

Vis-à-vis de Tree K-SVD, l'intérêt du "cerf-volant" vient de la fermeture des branches de l'arbre. Cela permet d'éviter de multiplier le nombre de dictionnaires une fois un certain niveau atteint. En refermant l'arbre avant d'atteindre un niveau trop vide et en apprenant à la place un unique dictionnaire par niveau, on évite le problème d'*overfitting*

et de perte d'efficacité des niveaux profonds de l'arbre. L'unique dictionnaire appris par niveau est alors plus général et complet.

### 3.3.4 Expérimentations

La structure en "cerf-volant" permet d'éviter la perte d'efficacité des niveaux profonds de l'arbre en élaguant les branches de l'arbre à un niveau donné et en les prolongeant par une unique branche, la "traîne du cerf-volant", permettant de ne pas limiter le nombre d'atomes pouvant être sélectionnés dans la structure (Fig. 3.7). Les résultats de la structure en "cerf-volant" sont ainsi similaires à ceux de l'arbre Tree K-SVD au début puis les dépassent nettement lorsque le nombre d'atomes dans la représentation augmente. Ils restent de plus supérieurs aux résultats du dictionnaire "plat" K-SVD. Pour l'image *New York*, on s'aperçoit qu'il est plus efficace de refermer les branches après 2 niveaux qu'après 3 (pour K=256) ou 4 niveaux (pour K=64).




(a) K = 64 (b) K = 256

FIGURE 3.7 – Tests PSNR-parcimonie de la structure en "cerf-volant" pour K=64 (a) et K=256 (b) sur *New York*. Le chiffre entre parenthèses indique le niveau après lequel l'ensemble des branches est refermé au sein de la structure en "cerf-volant".

Cependant, en testant la structure en "cerf-volant" sur une seconde image particulièrement homogène (Fig. 3.9), l'image *Désert* (Fig. 3.8) (image 8 bits en niveaux de gris de $2400 \times 2400$ pixels), on peut voir qu'il est plus intéressant pour cette image que la structure soit refermée après 4 niveaux pour K=64 (3 pour K=256) plutôt que 2. En effet, l'image de désert, homogène, est presque exclusivement décomposée sur les branches issues d'un atome quasiment constant au premier niveau de la structure. Cet atome étant très populaire au premier niveau, de nombreux vecteurs de résidus se dirigent à l'apprentissage vers ses branches filles de sorte que les dictionnaires de ces branches sont plus nombreux et perdent en efficacité plus tard que dans les autres branches de la structure, lorsque celle-ci est refermée après 3 ou 4 niveaux. *Désert* tire

ainsi partie d'une structure davantage développée, contrairement à *New York*. Le niveau optimal de fermeture des branches est donc différent selon l'image test, ce qui est problématique étant donné que l'image test est inconnue lors de l'apprentissage.

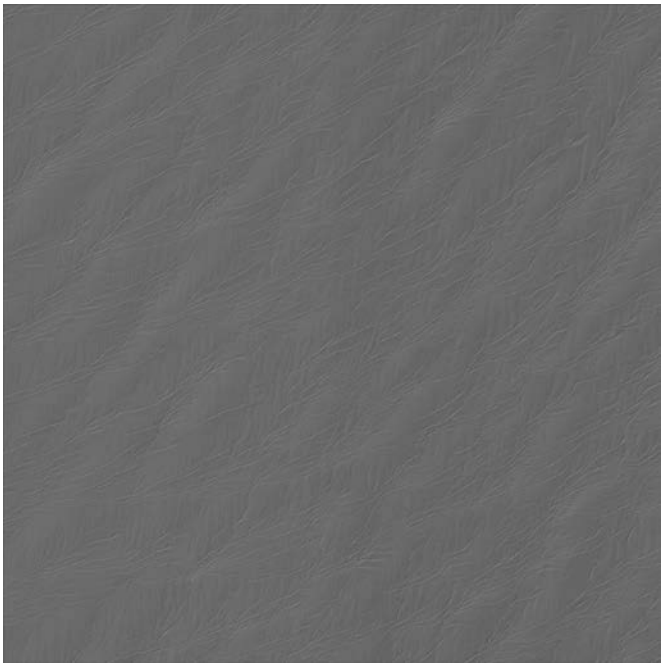
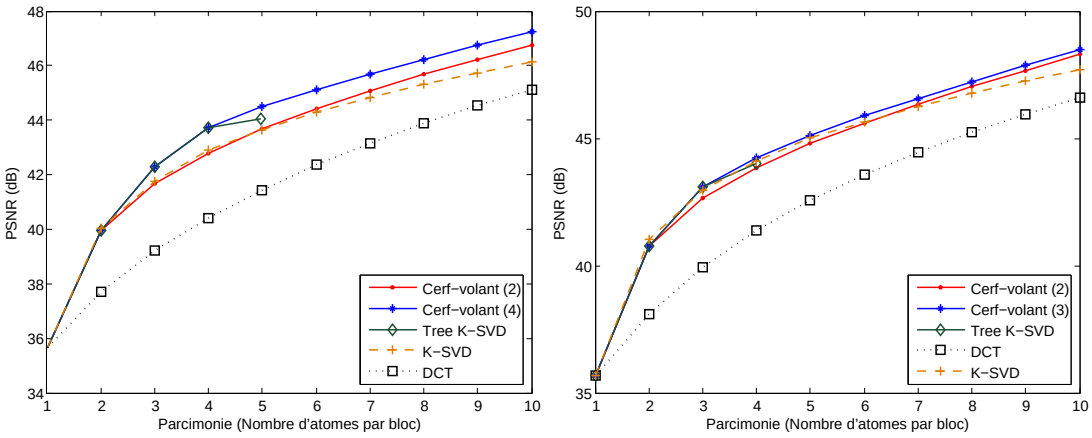


FIGURE 3.8 – Image test : *Désert* (2400x2400) (FTM de 0.35).

### 3.3.5 Limitations

La limitation majeure de la structure en "cerf-volant" tient dans le fait d'imposer un niveau de fermeture commun à toutes les branches. En effet, des branches très populaires, c'est-à-dire où se dirigent beaucoup de vecteurs d'apprentissage, peuvent profiter d'être davantage développées, tandis que d'autres branches nécessitent d'être rapidement élaguées. De plus, ce niveau de fermeture représente un paramètre de l'algorithme dont le réglage diffère selon l'image test à représenter, comme nous avons pu le voir grâce aux expérimentations du paragraphe précédent. Ainsi, il serait intéressant de rendre ce paramètre de fermeture variable selon les branches et de le déterminer de manière automatique de telle sorte qu'il soit adapté à toute image test.




(a) K = 64 (b) K = 256

FIGURE 3.9 – Tests PSNR-parcimonie de la structure en "cerf-volant" pour K=64 (a) et K=256 (b) sur *Désert*. Le chiffre entre parenthèses indique le niveau après lequel l'ensemble des branches est refermé au sein de la structure en "cerf-volant".

## 3.4 La Structure Adaptative

La Structure Adaptative cherche à corriger les limitations de la structure en arbre Tree K-SVD et de la structure en "cerf-volant". Ses objectifs sont donc d'éviter les problèmes d'inefficacité des niveaux profonds de l'arbre comportant de nombreux dictionnaires incomplets, voire vides, responsables de l'arrêt des branches, de façon à pouvoir apprendre une structure sur davantage de niveaux, et de s'affranchir du paramètre de fermeture commun à toutes les branches de la structure en "cerf-volant".

Pour cela, la Structure Adaptative (Fig. 3.10), dont la structure s'adapte automatiquement aux vecteurs d'entraînement lors de l'apprentissage, a été créée [AMGL13b]. Cette structure fonctionne comme une structure en arbre, mais dont les branches sont progressivement élaguées, selon leur popularité, pour être ensuite tour à tour fusionnées avec une unique branche commune.

Cette structure permet d'apprendre davantage de niveaux que Tree K-SVD tout en conservant un nombre d'atomes global plus raisonnable. L'idée de refermer les branches de l'arbre, émise pour la structure en "cerf-volant", est ici reprise. Mais plutôt que de refermer l'ensemble des branches à un niveau donné commun, chaque branche de la Structure Adaptative est refermée automatiquement indépendamment des autres branches sur un critère de popularité de la branche. On définit la popularité d'une branche à un niveau donné par le nombre de vecteurs de résidus se retrouvant dans cette branche à ce niveau, c'est-à-dire le nombre de vecteurs d'apprentissage à ce niveau donné permettant d'apprendre le dictionnaire au niveau suivant. Il est décidé de refermer une branche lorsque celle-ci ne comporte plus assez de vecteurs d'apprentissage pour apprendre un dictionnaire complet, c'est-à-dire lorsque le nombre de vecteurs

d'apprentissage devient strictement inférieur à $K$, de telle sorte qu'aucun dictionnaire incomplet ne soit créé au sein de la structure.

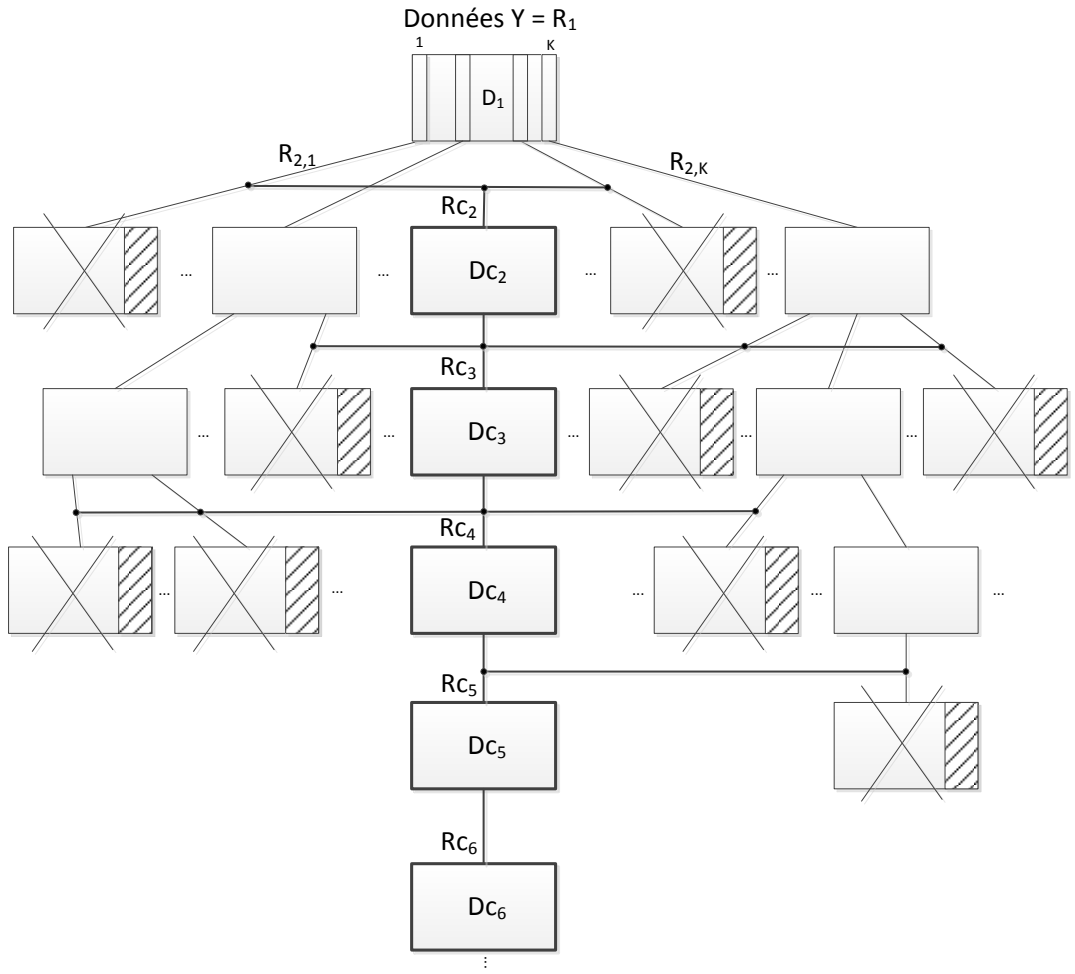



FIGURE 3.10 – La Structure Adaptative. Les dictionnaires rayés représentent les dictionnaires incomplets non créés, les vecteurs de résidus pour leur apprentissage étant trop peu nombreux. Ces derniers sont donc concaténés à chaque niveau $i$ dans $Rc_i$ pour apprendre le dictionnaire commun $Dc_i$.

### 3.4.1 Apprentissage de la structure

La Structure Adaptative est apprise de haut en bas, niveau par niveau. Comme pour les deux structures précédentes, l'algorithme K-SVD est utilisé avec une parcimonie de 1 atome pour apprendre chaque dictionnaire de $K$ atomes au sein de la structure, et l'algorithme MP est appliqué avec une parcimonie de 1 atome également afin de

représenter chaque vecteur d'apprentissage d'un dictionnaire donné par un atome de ce dictionnaire pour en calculer les résidus afin d'apprendre le niveau suivant.

Le dictionnaire au premier niveau $D_1$ est classiquement appris sur l'ensemble des vecteurs d'apprentissage $Y$. Puis les résidus $R_2$ sont calculés et partitionnés en $K$ ensembles de résidus, un par atome de $D_1$. Pour chaque ensemble $R_{2,k}, k \in [1, ..., K]$, si le nombre de vecteurs d'entraînement qu'il contient est suffisant pour apprendre un dictionnaire complet, c'est-à-dire si ce nombre est supérieur ou égal à $K$ [1], alors le dictionnaire correspondant au troisième niveau est appris de manière classique. Dans le cas contraire, le dictionnaire, qui aurait été incomplet, n'est pas créé, la branche est donc élaguée, et les vecteurs de résidus dans $R_{2,k}$ sont sauvegardés. Cette procédure est appliquée pour chaque ensemble de résidus $R_{2,k}$ au deuxième niveau. Ensuite, tous les résidus sauvegardés lors de l'apprentissage de ce deuxième niveau, qui n'ont donc pas servis à apprendre de dictionnaire, sont concaténés en $Rc_2$ pour apprendre un dictionnaire commun et plus général $Dc_2$, le premier dictionnaire de la branche commune de la structure. A chaque niveau $i$, les résidus en nombre insuffisant pour apprendre un dictionnaire sont donc sauvegardés et concaténés ensembles, ainsi qu'avec les résidus issus du dictionnaire commun du niveau précédent, afin de former un ensemble de vecteurs de résidus $Rc_i$ sur lequel est appris le dictionnaire commun $Dc_i$ du niveau courant. Ce dictionnaire $Dc_i$ représente alors le dictionnaire fils de tous les atomes du niveau précédent dont les branches ont été élaguées à ce niveau. Cette procédure est répétée le nombre de niveaux souhaités.

Avec cette méthode d'apprentissage, les branches très populaires, c'est-à-dire les branches vers lesquelles beaucoup de vecteurs de résidus se dirigent, sont davantage développées que les branches peu populaires, qui sont rapidement élaguées et voient leurs résidus concaténés aux résidus communs afin désormais d'apprendre la branche commune de la structure. Si la structure est apprise sur suffisamment de niveaux, toutes les branches sont donc progressivement élaguées pour que finalement, la structure ne contienne plus qu'une branche (la branche commune) et donc un dictionnaire par niveau. Ainsi, durant la procédure d'apprentissage, la structure s'adapte automatiquement aux vecteurs d'apprentissage, et notamment à leur nombre, en développant plus ou moins ses différentes branches. La structure sera par exemple peu développée pour un ensemble d'apprentissage relativement restreint, de façon à limiter l'*overfitting*, tandis qu'un grand nombre de vecteurs d'apprentissage pourra conduire à une structure comprenant davantage de dictionnaires afin d'améliorer l'apprentissage sur ces nombreuses données.

### 3.4.2 Décomposition au sein de la structure

La décomposition d'un vecteur test au sein de la Structure Adaptative est toujours réalisée en sélectionnant 1 atome par niveau, à l'aide de l'algorithme MP, le

1. Un paramètre permet de régler le nombre minimum de vecteurs d'apprentissage requis pour apprendre un dictionnaire. Il prend par défaut dans les expérimentations la valeur $K$ correspondant à la taille de chaque dictionnaire afin que la structure ne contienne pas de dictionnaire incomplet et qu'ainsi tous les dictionnaires de la structure soient de même taille.

dictionnaire à utiliser à chaque niveau étant indiqué par l'atome sélectionné au niveau précédent. La différence par rapport à la structure en arbre vient lorsque la fin d'une branche est atteinte. Là où la décomposition s'arrêtait pour Tree K-SVD, elle est ici poursuivie au sein de la branche commune de la Structure Adaptative, ce qui permet de pouvoir sélectionner davantage d'atomes pour atteindre une meilleure qualité de représentation. Par rapport à la structure en "cerf-volant" où la décomposition continuait sur la "traîne" à un niveau précis, la décomposition au sein de la Structure Adaptative rejoint la branche commune plus ou moins vite selon la branche parcourue. Une fois la branche commune rejointe, la décomposition continue sur cette branche jusqu'à ce que la parcimonie souhaitée, ou le dernier niveau de la structure, soit atteint.

### 3.4.3 Intérêts par rapport aux structures en arbre et en "cerf-volant"

La Structure Adaptative présente d'abord les mêmes avantages que la structure en arbre et en "cerf-volant". Composée de petits dictionnaires de $K$ atomes, elle permet de contenir globalement de nombreux atomes tout en conservant un coût de codage de chaque indice et une complexité de décomposition équivalents à ceux d'un dictionnaire "plat" de $K$ atomes. Elle est également scalable en parcimonie, puisque adaptée à des décompositions pour plusieurs valeurs de parcimonie.

Par rapport à la structure en arbre Tree K-SVD, elle permet d'éviter le problème des niveaux profonds trop vides en élaguant les branches avant d'atteindre un dictionnaire incomplet, tout en apprenant une structure sur davantage de niveaux grâce à la branche commune afin de ne pas limiter le nombre d'atomes sélectionnés.

Enfin, vis-à-vis de la structure en "cerf-volant", les branches sont progressivement élaguées et refermées sur la branche commune, le niveau de fermeture s'ajustant automatiquement pour chaque branche selon sa popularité. La structure s'adapte ainsi automatiquement selon les données d'apprentissage.

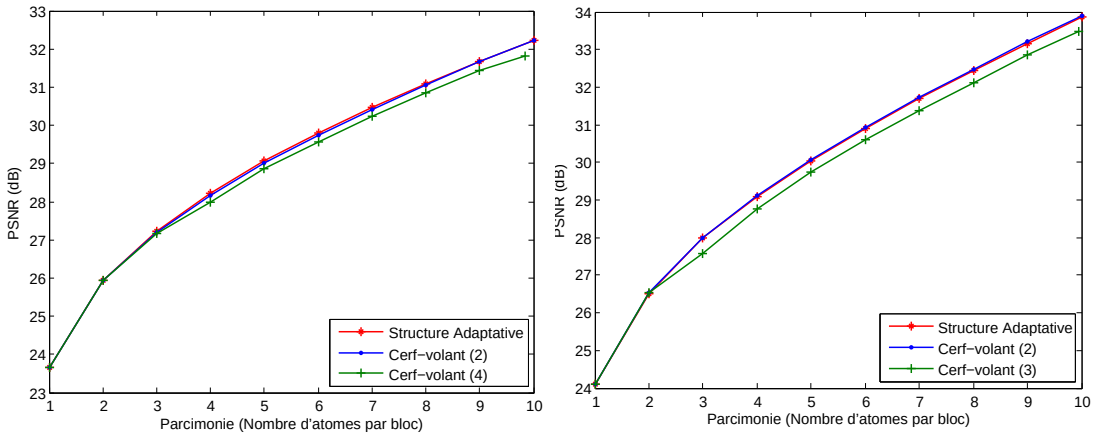
### 3.4.4 Expérimentations

La Structure Adaptative permet comme la structure en "cerf-volant" d'éviter la perte d'efficacité des niveaux profonds du Tree K-SVD en optant pour la même stratégie d'élagage des branches puis de prolongation par une branche commune. Elle présente donc tout d'abord des résultats similaires à Tree K-SVD, puis les surpasse (Fig. 3.13).

Le fait de refermer les branches progressivement selon leur popularité, plutôt que d'imposer un niveau de fermeture commun, permet d'obtenir des résultats similaires à la structure en "cerf-volant" refermée après le nombre de niveaux optimal selon l'image test (Fig. 3.11 et Fig. 3.12). La Structure Adaptative s'adapte ainsi afin de se dispenser de ce paramètre dépendant de l'image test, sans pour autant détériorer les résultats.

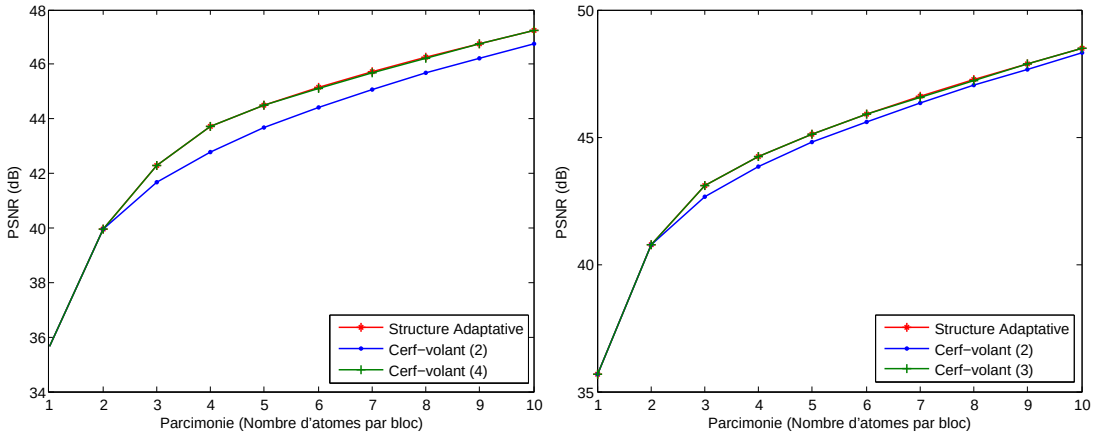
Par rapport à une structure en branche ne comportant qu'un seul dictionnaire par niveau, également appris avec K-SVD pour une parcimonie de 1 atome, la Structure Adaptative tire partie de sa structure non rigide pouvant comporter davantage de dictionnaires pour présenter des résultats légèrement supérieurs, en particulier pour K=64 et un faible nombre d'atomes sélectionnés (Fig. 3.13). Sur une image comme *Désert*,

plus proche de certaines images d'apprentissage, l'écart entre la Structure Adaptative et la structure en branche est plus prononcé (Fig. 3.14), en particulier pour K=64.




(a) K = 64 (b) K = 256

FIGURE 3.11 – Comparaison entre la Structure Adaptative et la structure en "cerf-volant" pour K=64 (a) et K=256 (b) sur *New York*.

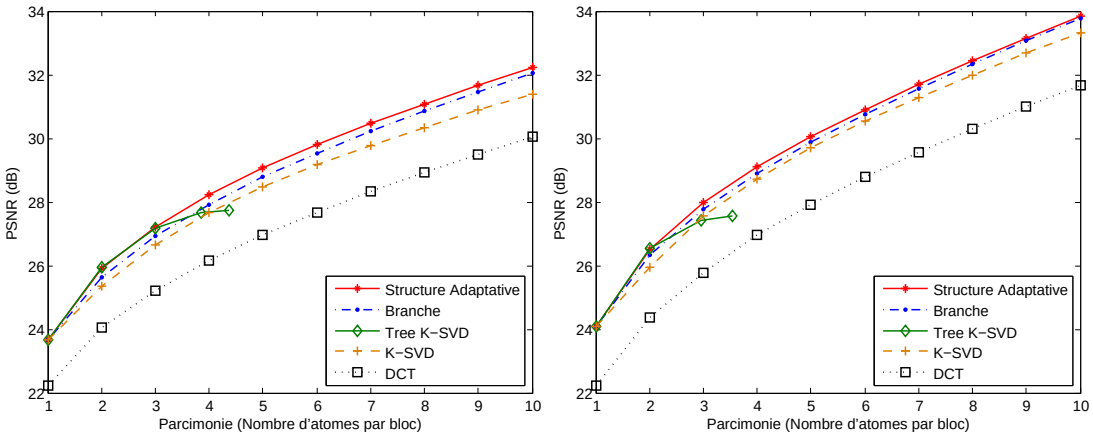



(a) K = 64 (b) K = 256

FIGURE 3.12 – Comparaison entre la Structure Adaptative et la structure en "cerf-volant" pour K=64 (a) et K=256 (b) sur *Désert*.

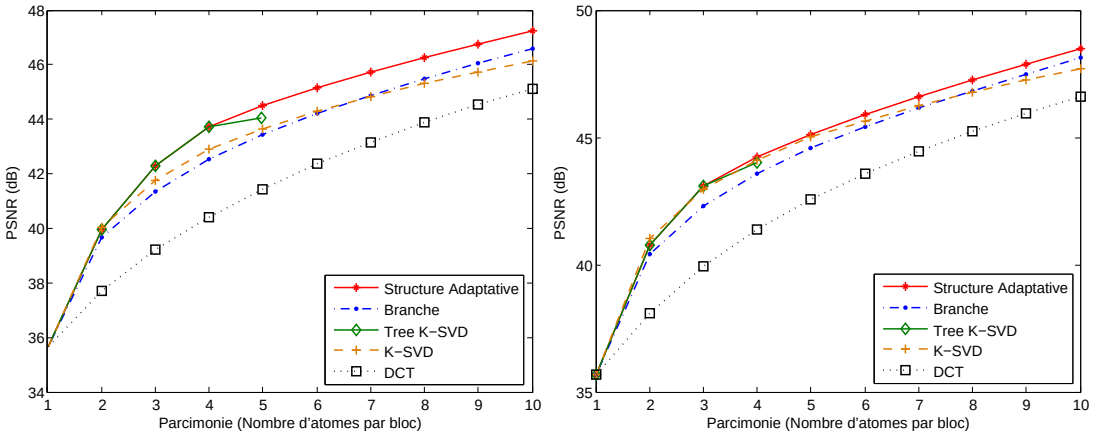
### 3.4.5 Limitation

La limitation principale de la Structure Adaptative est également ce qui fait sa force : le nombre conséquent de dictionnaires qu'elle contient. En effet, comme pour les autres




(a) K = 64 (b) K = 256

FIGURE 3.13 – Tests PSNR-parcimonie de la Structure Adaptative pour K=64 (a) et K=256 (b) sur *New York*.




(a) K = 64 (b) K = 256

FIGURE 3.14 – Tests PSNR-parcimonie de la Structure Adaptative pour K=64 (a) et K=256 (b) sur *Désert*.

structures que sont Tree K-SVD et la structure en "cerf-volant", ces nombreux dictionnaires nécessitent d'être stockés. C'est pourquoi, dans le cas où cela représenterait une véritable contrainte, un paramètre de l'algorithme apprenant la Structure Adaptative permet de régler le nombre de vecteurs d'apprentissage minimum requis pour apprendre un dictionnaire. Si ce nombre n'est pas atteint, alors le dictionnaire n'est pas créé et la branche est élaguée. Par défaut à $K$ (nombre d'atomes dans chaque dictionnaire) dans les expérimentations afin de ne pas créer de dictionnaire incomplet, augmenter la

valeur de ce paramètre permet d'élaguer plus tôt les différentes branches afin de limiter la taille globale de la structure, au risque de diminuer ses performances. En faisant tendre ce paramètre vers l'infini, on apprendrait ainsi une structure en branche.

## 3.5 Traitement séparé de la moyenne des patchs

Il est fréquent de voir dans la littérature la moyenne de chaque vecteur d'apprentissage, et donc de chaque vecteur test, retirée. Le coefficient DC d'une transformation contient fréquemment une part significative de l'énergie totale d'un signal. Et comme il existe en général une forte corrélation entre les coefficients DC de deux blocs voisins, il peut être intéressant de traiter ce coefficient différemment des autres coefficients. Dans JPEG par exemple, un encodage spécifique est appliqué aux coefficients DC des différents blocs. Plusieurs méthodes présentées dans le premier chapitre (par exemple [XLLZ14], [ZGK11]) traitent également les valeurs de moyenne des patchs séparément des coefficients et indices.

Cependant, dans les expérimentations présentées lors de ce chapitre, la moyenne des patchs, aussi bien d'apprentissage que de test, a été conservée. Nous allons voir que ce choix rend les structures déséquilibrées, ce qui nous a amené à corriger cela en retirant la moyenne de chaque patch lors de nouvelles expérimentations.

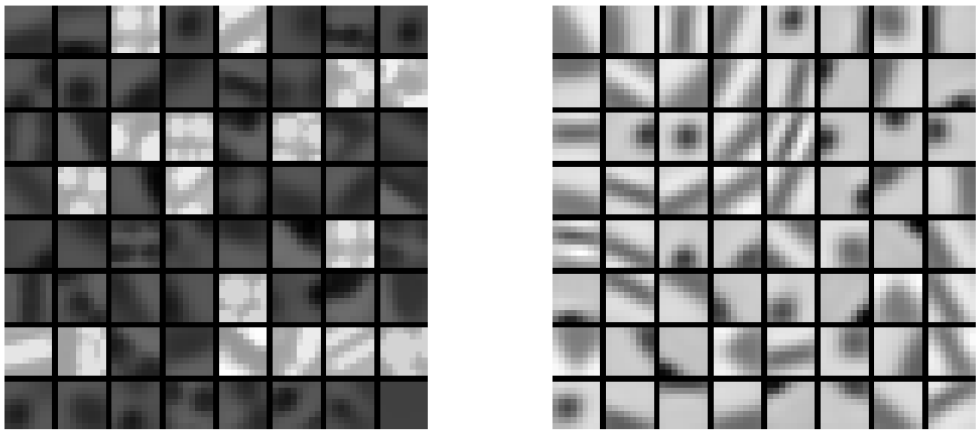
### 3.5.1 Pourquoi retirer la moyenne des patchs ?

En apprenant les dictionnaires sur les patchs d'entraînement (toujours sur la même base d'images d'apprentissage, présentée en Annexe 1) sans en retirer la valeur moyenne, on constate au premier niveau des différentes structures, appris avec K-SVD pour une parcimonie de 1 atome, un atome presque constant (Fig. 3.15 (a), atome 64). On note que les atomes représentés, les valeurs du dictionnaire étant ramenées entre 0 et 255 pour l'affichage, sont soit clairs soit foncés. Les atomes clairs correspondent à des atomes dont toutes les valeurs sont positives tandis que les atomes foncés correspondent à des atomes dont toutes les valeurs sont négatives. L'absence de variation de signe au sein des atomes s'explique par le fait que l'apprentissage est effectué seulement sur des blocs image dont les valeurs ont toutes le même signe, en l'occurrence positif.

En observant la popularité des différents atomes de ce dictionnaire au premier niveau, en appliquant l'algorithme MP sur les vecteurs d'entraînement et le dictionnaire avec une parcimonie de 1 atome, on s'aperçoit alors que cet atome quasiment constant est très majoritairement choisi (Fig. 3.16 (a)). De telle sorte que les structures de dictionnaires sont par la suite très inégalement réparties du fait de la popularité de la branche issue de l'atome constant vis-à-vis des autres branches.

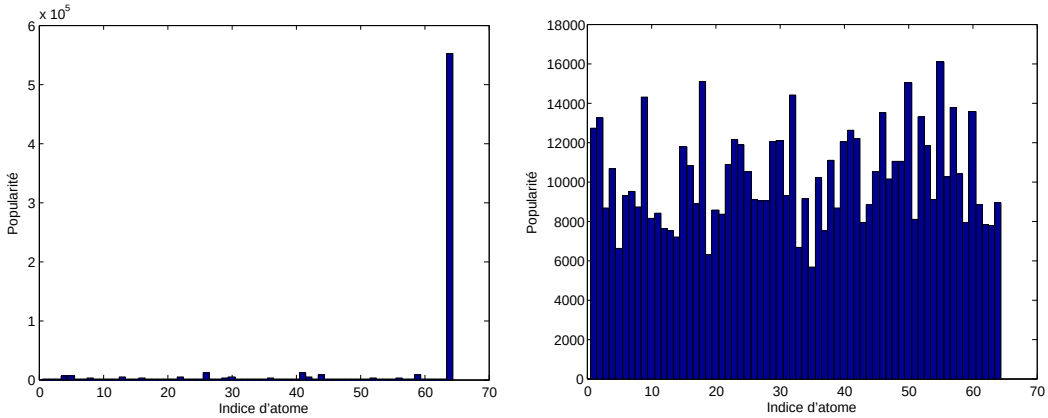
De plus, au sein de l'algorithme K-SVD, des atomes non ou peu utilisés lors d'une itération sont remplacés par les vecteurs d'apprentissage les plus mal représentés. Ainsi, lorsque la valeur de parcimonie est faible (ici 1 atome), et encore davantage lorsque le dictionnaire comporte de nombreux atomes, la très faible utilisation des autres atomes que l'atome constant peut conduire à de nombreux remplacements et des problèmes de convergence du K-SVD pour apprendre ce premier niveau (Fig. 3.17 (a)).

Enfin, si cet atome constant est majoritairement choisi lors des décompositions comme premier atome, c'est d'abord pour représenter la moyenne du bloc. Afin de ne pas coder l'indice correspondant, on décide donc de retirer la moyenne des patchs avant la décomposition et de la coder séparément. A l'apprentissage, la moyenne des vecteurs d'entraînement est également retirée avant d'apprendre les dictionnaires.



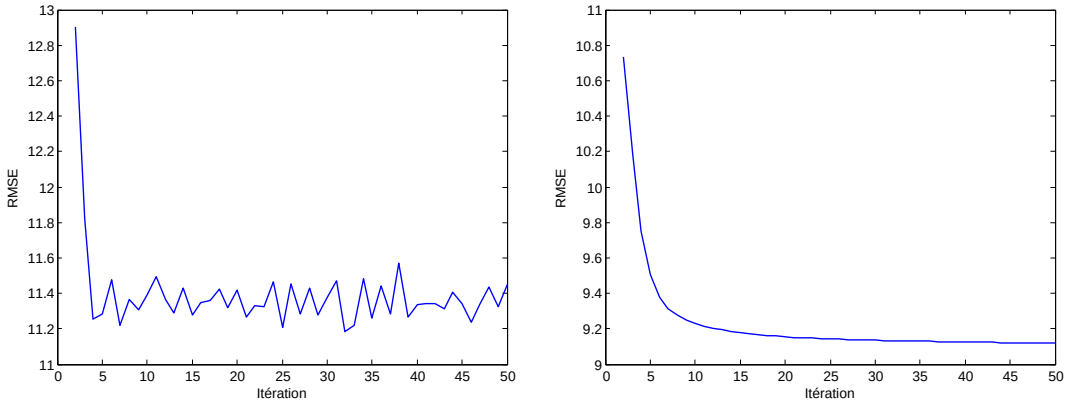
(a) Patchs avec la moyenne (b) Patchs sans la moyenne

FIGURE 3.15 – Dictionnaire (K=64) au premier niveau appris sur les patchs avec (a) ou sans leur valeur moyenne (b). Les atomes du dictionnaire sont rangés en colonne. Chaque atome, rangé en colonnes, est représenté sous la forme d'un bloc.




(a) Patchs avec la moyenne (b) Patchs sans la moyenne

FIGURE 3.16 – Popularités des atomes au premier niveau vis-à-vis des patchs d'apprentissage avec (a) ou sans leur valeur moyenne (b) (K=64).




(a) Patchs avec la moyenne

(b) Patchs sans la moyenne

FIGURE 3.17 – Convergence de l'erreur (RMSE) de représentation des patchs d'apprentissage en fonction des itérations de l'algorithme K-SVD pour une parcimonie de 1 atome, en apprenant sur les patchs avec (a) ou sans leur valeur moyenne (b) (K=64).

### 3.5.2 Expérimentations

En extrayant la moyenne des patchs d'entraînement avant d'apprendre les dictionnaires, aucun atome constant n'est appris au premier niveau (Fig. 3.15 (b)) et la popularité des différents atomes s'en trouve plus équilibrée (Fig. 3.16 (b)), ce qui permet d'apprendre des structures elles aussi plus équilibrées. Cela permet de plus de corriger les problèmes de convergence de K-SVD pour une parcimonie de 1 atome au premier niveau (Fig. 3.17 (b)).

La Structure Adaptative est de nouveau apprise pour K=64 et K=256. La Figure 3.18 présente certains dictionnaires de la structure pour K=64 aux niveaux 1, 2 et 3. On remarque que les dictionnaires fils d'atomes directionnels au premier niveau présentent globalement la même direction principale, les structures s'atténuant par la suite au fur et à mesure des niveaux. Le dictionnaire commun au niveau 3 (Fig. 3.19), appris sur l'ensemble des résidus des branches élaguées après le deuxième niveau, ne semble pas présenter de structure particulière.

En observant les structures plus en détail, on remarque que la Structure Adaptative, apprise sur 20 niveaux, possède des branches davantage développées pour $K = 64$ que pour $K = 256$. En effet, la structure comporte 4149 dictionnaires, soit 265536 atomes, pour $K = 64$, et 278 dictionnaires, soit 71168 atomes, pour $K = 256$. Pour $K = 64$, les branches sont élaguées après le niveau 2 pour les premières et après le niveau 4 pour les dernières. A partir du niveau 5, il n'y a donc plus qu'un dictionnaire (le dictionnaire commun) par niveau. Pour $K = 256$, les premières branches sont également élaguées après le niveau 2, mais les dernières le sont après le niveau 3 de sorte qu'il n'y a plus qu'un dictionnaire par niveau à partir du niveau 4.

Suite au retrait de la moyenne des patchs d'apprentissage, les autres structures de dictionnaires, ainsi que les dictionnaires "plats" sont également de nouveau appris.

Les tests sont effectués sur 3 images distinctes : *Désert*, *New York* et *Hambourg*, des images 8 bits en niveaux de gris de $2400 \times 2400$ pixels.

L'image *Désert* (Fig. 3.8) est une image très homogène comportant très peu de structures. Elle est proche d'images de désert présentes dans la base d'apprentissage.

L'image *New York* (Fig. 3.3) est plus complexe et présente à la fois des zones homogènes comme des zones texturées de ville. Moins proche d'images de la base d'apprentissage que l'image *Désert*, d'autres images de la ville de New York sont toutefois présentes dans la base, représentant des zones voisines. L'image test représente d'ailleurs sur sa partie inférieure (les 200 dernières lignes) une zone commune avec une image d'apprentissage. Même si les valeurs ne sont pas identiques aux valeurs de pixels près, les blocs de test et d'apprentissage créées sur cette zone sont très similaires (voir l'impact au codage sur la parcimonie par bloc Fig. 4.4). Cette zone étant relativement restreinte (8.3% de l'image), son influence sur les résultats reste mesurée, mais elle fait de *New York* une image assez proche des données d'apprentissage.

Enfin, l'image *Hambourg* (Fig. 3.20) est l'image test la plus éloignée des images d'apprentissage. Aucune autre image de Hambourg n'est présente dans la base d'apprentissage. Elle représente des environnements variés tels que de la ville, des champs, de l'eau, ou encore de la forêt.

Ces trois images tests étant plus ou moins proches des données d'apprentissage, cela va nous permettre d'observer les écarts entre les différentes structures de dictionnaires selon l'éloignement entre l'image test et les images d'apprentissage.

En réalisant tout d'abord les tests sur l'image *New York* (Fig. 3.21), on constate de nouveau l'écart entre le dictionnaire DCT prédéfini et les dictionnaires appris.

Ensuite, K-SVD [AEB06] et Sparse K-SVD [RZE10] (la toolbox fournie par l'auteur [Rub] est utilisée) donnent des résultats très proches, avec un apprentissage plus rapide pour Sparse K-SVD. En diminuant la parcimonie du dictionnaire appris jusqu'à 16 valeurs non nulles par colonne, les résultats deviennent légèrement inférieurs à ceux obtenus avec une parcimonie du dictionnaire de 32.

Tree K-SVD et TSITD possèdent tous les deux une structure similaire en arbre et ont été appris sur 4 niveaux pour K=64 et 3 niveaux pour K=256. Efficaces sur les premiers niveaux, ils sont rapidement dépassés par K-SVD. Tree K-SVD est légèrement plus efficace sur les niveaux plus profonds (3 et 4 pour K=64 et 3 pour K=256) grâce à sa stratégie de copie des vecteurs d'apprentissage, lorsque ces derniers ne sont pas assez nombreux pour apprendre un dictionnaire complet. Pour TSITD, la stratégie utilisée est dans ce cas là d'apprendre un plus petit dictionnaire que dans le Tree K-SVD. Même si ce dictionnaire est appris, la limitation de sa taille pénalise les résultats par rapport à Tree K-SVD. L'utilisation de la SAAN pour la décomposition sur l'arbre appris par Tree K-SVD améliore fortement les résultats. Mais la comparaison n'est alors plus à coût de codage des indices et à complexité équivalents. De plus, les niveaux profonds inefficaces de l'arbre sont inévitablement atteints et Tree K-SVD SAAN est rattrapé

par K-SVD.

La Structure Adaptative et la structure en "cerf-volant", lorsque le niveau d'élagage des branches est bien choisi, permettent de compenser les limitations de la structure en arbre et offrent ainsi de meilleurs résultats que K-SVD et Sparse K-SVD. Les résultats de la structure en branche, structure similaire au BITD ne comportant qu'un seul dictionnaire par niveau mais appris avec K-SVD pour une parcimonie de 1 atome, sont proches de ceux de la Structure Adaptative, particulièrement pour K=256, mais légèrement inférieurs pour K=64, notamment lorsque le nombre d'atomes par bloc est faible. On remarque ainsi que le fait d'extraire la moyenne des patchs rapproche les résultats de la structure en branche de ceux de la Structure Adaptative. Cette dernière s'adapte donc mieux aux différentes situations grâce à sa structure adaptative. Les figures (c) et (d) permettent de comparer les différentes structures en utilisant le même algorithme pour apprendre chaque dictionnaire au sein des différentes structures : K-SVD.

On note enfin que la structure ITAD offre de meilleurs résultats que la Structure Adaptative pour K=64 et similaires pour K=256. Outre l'utilisation de matrices d'alignement permettant d'aligner les résidus à chaque niveau, cet écart s'explique en partie par un algorithme d'apprentissage de chaque dictionnaire au sein de la structure légèrement différent. En effet, même si l'approche est similaire à K-SVD avec une parcimonie de 1 atome, il existe quelques différences. Plusieurs initialisations, en sélectionnant des vecteurs d'apprentissage, sont par exemple essayées sur quelques itérations avant de choisir la plus performante. Suite à l'étape de mise-à-jour du dictionnaire, une étape de re-classification des données d'apprentissage sur les différents atomes peut également être appliquée. Du point de vue complexité, même si ITAD bénéficie d'une réduction de la dimension du dictionnaire à chaque niveau, les multiplications par les matrices d'alignement rendent la décomposition plus complexe qu'avec la Structure Adaptative, à moins de sélectionner un nombre d'atomes important (en particulier pour K=64). La comparaison avec ITAD n'est donc pas effectuée à complexité égale.

En réalisant les mêmes tests sur l'image *Désert*, proche de certaines images d'apprentissage, on remarque que cela profite aux structures les plus développées (Fig. 3.22). Tree K-SVD donne ainsi de meilleurs résultats, bien que souffrant toujours des mêmes limitations. Pour K=64, la structure en "cerf-volant" élaguée après 4 niveaux donne de meilleurs résultats sur les premiers niveaux que celle élaguée après 2 niveaux, mais est ensuite rattrapée, trop pénalisée par un niveau 4 probablement de trop. Élaguer les branches après 3 niveaux aurait sans doute été le plus efficace pour cette image test. La Structure Adaptative permet d'éviter ce paramètre et reste supérieure aux deux structures en "cerf-volant". Dans ce cas où l'image test est plus proche de certaines images d'apprentissage, la Structure Adaptative montre un net avantage sur la structure en branche pour K=64. Les résultats sont en revanche similaires pour des dictionnaires plus larges (K=256). Enfin, les résultats de la Structure Adaptative et de ITAD sont plus proches pour cette image test.

Au contraire, réaliser les tests sur l'image *Hambourg*, s'éloignant davantage des images d'apprentissage, profite à la structure en branche, dont chaque dictionnaire est plus général car appris sur l'ensemble des résidus. Celle-ci présente alors pour K=256 des résultats légèrement supérieurs à la Structure Adaptative (Fig. 3.23) lorsque le nombre d'atomes sélectionnés augmente.

Ainsi, si l'intérêt de la Structure Adaptative par rapport aux dictionnaires "plats" K-SVD ou Sparse K-SVD est bien marqué, il est cependant plus nuancé par rapport à la structure en branche, présentant des résultats en général assez proches. Il dépend alors de la taille des dictionnaires dans les structures et de la proximité de l'image test avec les images d'apprentissage. La Structure Adaptative tire en effet profit de sa large structure par rapport à la structure en branche qui est figée lorsque l'image test est très proche des images d'apprentissage, en particulier lorsque la taille des dictionnaires est limitée (K=64) et que le nombre d'atomes sélectionnés reste faible. Si l'image test s'éloigne davantage des images d'apprentissage, la structure en branche peut alors devenir plus intéressante de part la généricité plus importante de ces dictionnaires, en particulier lorsque la taille des dictionnaires est importante (K=256). La Structure Adaptative reste toutefois efficace dans ce cas avec des résultats relativement proches.

Dans le but de tester l'importance de l'initialisation de chaque dictionnaire au sein de la Structure Adaptative, une initialisation par des données d'apprentissage est testée en remplacement d'un dictionnaire DCT (Fig. 3.24). Des vecteurs d'apprentissage sont alors sélectionnés de façon aléatoire afin de servir d'initialisation pour l'apprentissage réalisé par l'algorithme K-SVD avec une parcimonie de 1 atome. Deux Structures Adaptatives initialisées de la sorte ont été apprises afin d'être comparées à la Structure Adaptative initialisée par des dictionnaires DCT. En testant sur l'image *New York*, on remarque que les résultats obtenus sont peu sensibles à l'initialisation des dictionnaires. On arrive à la même conclusion en testant sur les images *Désert* et *Hambourg*.

### 3.5.3 Une décomposition de type OMP sur les dictionnaires structurés

Jusqu'à présent, la décomposition au sein des dictionnaires structurés pouvait être assimilée à une décomposition de type MP, mais en utilisant un nouveau dictionnaire à chaque itération. L'idée est ici d'adapter la décomposition OMP aux dictionnaires structurés afin d'améliorer la qualité de la reconstruction. Pour chaque bloc et comme précédemment, 1 atome est sélectionné par niveau, en cherchant l'atome dont la valeur absolue du produit scalaire avec le résidu courant est la plus importante. Cependant à chaque itération, une fois l'atome sélectionné, l'ensemble des coefficients précédemment calculés est de nouveau calculé afin de chercher les coefficients optimaux pour les atomes sélectionnés. Cela permet de rendre le résidu orthogonal à tous les atomes sélectionnés jusque là. Pour recalculer les coefficients, la même méthode que OMP est utilisée et fait donc intervenir le calcul de la pseudo-inverse des atomes sélectionnés jusqu'à l'itération courante. La décomposition de type OMP est donc plus complexe que la décomposition

de type MP mais est de complexité équivalente à l'application de OMP sur les dictionnaires "plats".

Appliquée pour la Structure Adaptative, sur les mêmes structures déjà apprises, la décomposition de type OMP permet d'améliorer la qualité de reconstruction par rapport à la décomposition de type MP (Fig. 3.21 - 3.23 (e)(f)). Associée à cette nouvelle décomposition, les résultats de la Structure Adaptative dépassent ainsi ceux d'ITAD, en particulier pour K=256 et plus légèrement pour K=64.

A noter que cette décomposition de type OMP peut également être appliquée à une structure en branche.

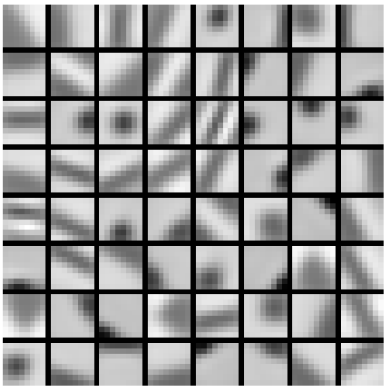
### 3.5.4 Le critère d'arrêt de la décomposition

Plutôt que de stopper la décomposition dès lors qu'un critère de parcimonie (critère orienté débit) est atteint, essayons désormais de fixer une erreur cible (critère orienté distorsion) à atteindre, pour chaque bloc de l'image test, comme critère d'arrêt. Pour cela, les mêmes Structure Adaptative et structure en branches sont utilisées, la décomposition (de type MP) par bloc se poursuit alors plus ou moins profondément dans la structure selon le bloc courant jusqu'à atteindre le critère d'erreur fixé. Pour K-SVD, un dictionnaire est appris par erreur cible et est testé pour le même critère d'erreur. La sélection des atomes est alors réalisée avec OMP et s'arrête lorsque l'erreur cible est atteinte.

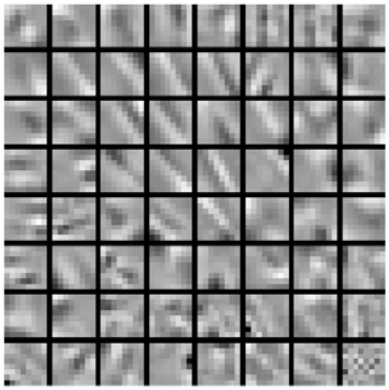
Ce critère d'arrêt permet ainsi de représenter les différents blocs avec une parcimonie variable selon le contenu de chaque bloc. En effet, un bloc homogène aura besoin de moins d'atomes qu'un bloc au contenu plus texturé pour atteindre une même erreur de représentation.

Les tests sont réalisés sur l'image *New York* (Fig. 3.25). Dans un premier temps, on observe une plus grande différence entre K-SVD et les structures de dictionnaires (Structure Adaptative et structure en branche), à l'avantage de ces dernières, en utilisant l'erreur comme critère d'arrêt plutôt que la parcimonie. Ensuite, il est clair qu'utiliser ce critère d'arrêt permet d'obtenir de bien meilleurs résultats que lorsque tous les blocs sont représentés pour une même parcimonie. En effet, les blocs ont besoin de plus ou moins d'atomes pour les représenter selon leur contenu plus ou moins complexe. Ainsi, un bloc homogène est correctement représenté avec très peu d'atomes, alors plutôt que d'ajouter de nouveaux atomes pour l'approximation de ce bloc, il est plus intéressant de les utiliser pour mieux représenter un bloc au contenu plus complexe. En répartissant mieux le nombre d'atomes utilisés dans la représentation entre les blocs, on atteint donc une meilleure qualité de représentation de l'image globale.

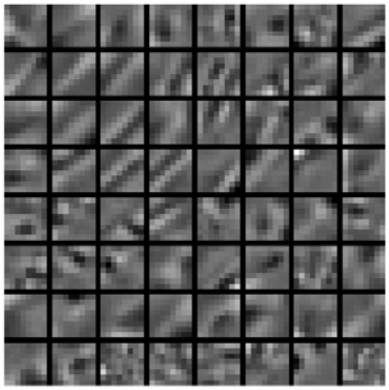
Il sera intéressant de retrouver cette parcimonie variable entre les blocs dans le cas du codage également, au chapitre suivant.



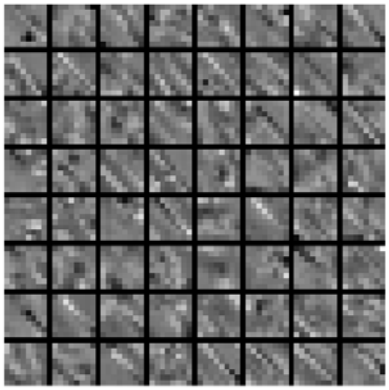
(a) Niveau 1



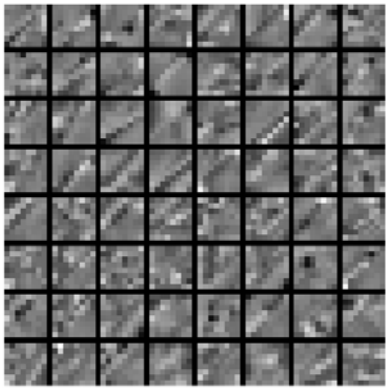
(b) Niveau 2 : (1)



(c) Niveau 2 : (18)



(d) Niveau 3 : (1,28)



(e) Niveau 3 : (18,10)

FIGURE 3.18 – Dictionnaires (K=64) de la Structure Adaptative. Les atomes de chaque dictionnaire sont rangés en colonne. Chaque atome, rangé en colonnes, est représenté sous la forme d'un bloc. Les nombres entre parenthèses indiquent les atomes pères aux niveaux précédents. Les dictionnaires de gauche correspondent à une branche et ceux de droite à une autre branche, toutes deux issues d'un atome différent du dictionnaire au premier niveau (a).

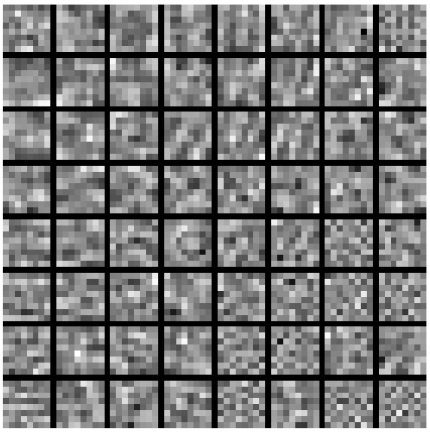


FIGURE 3.19 – Dictionnaire commun du niveau 3 de la Structure Adaptative (K=64). Les atomes du dictionnaire sont rangés en colonne. Chaque atome, rangé en colonnes, est représenté sous la forme d'un bloc.

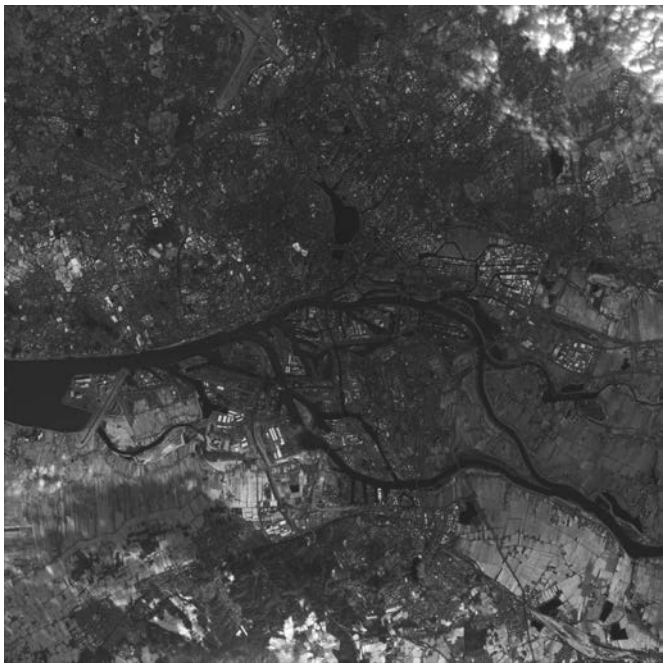
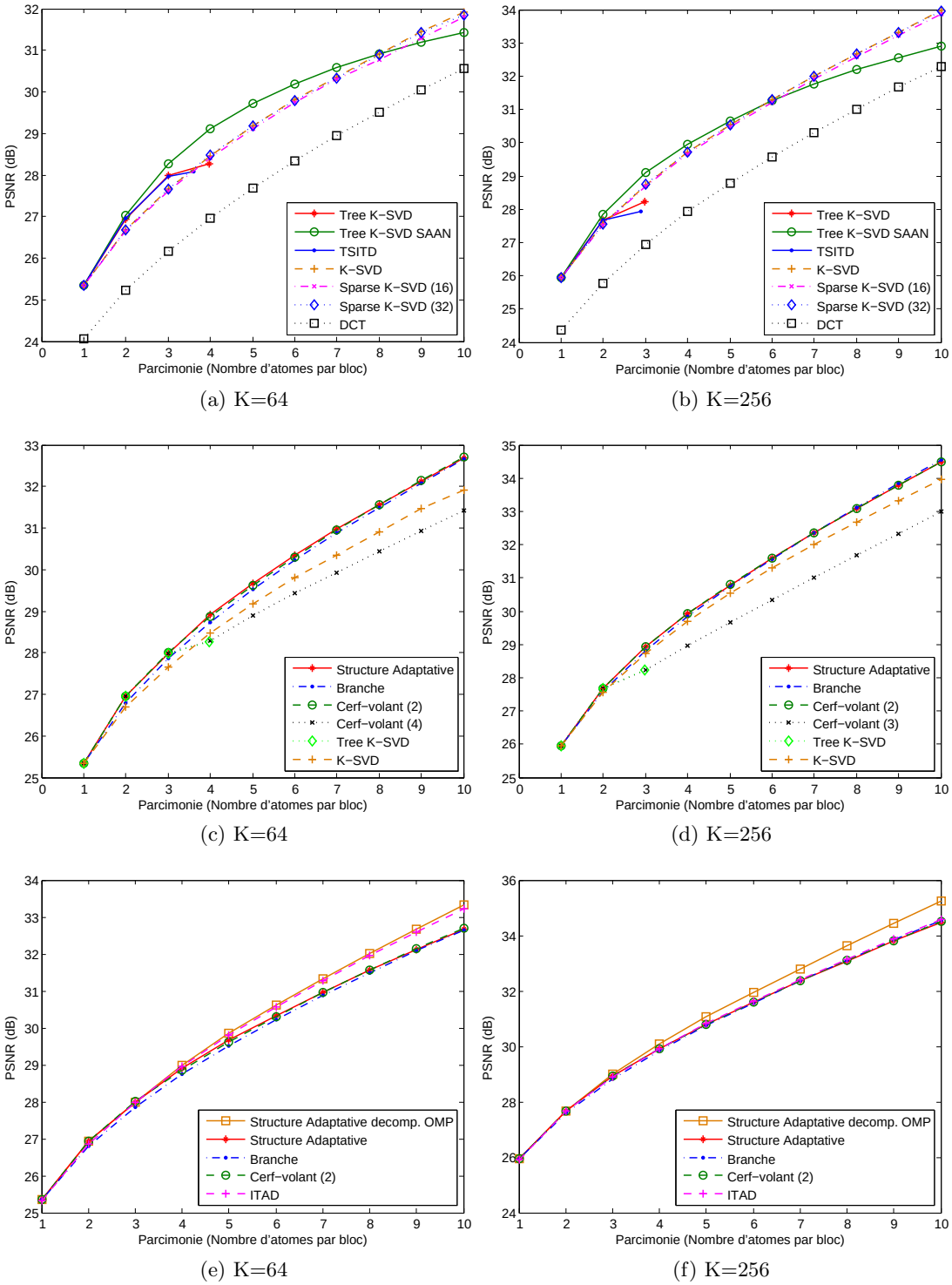


FIGURE 3.20 – Image test : *Hambourg* (2400x2400) (FTM de 0.36).

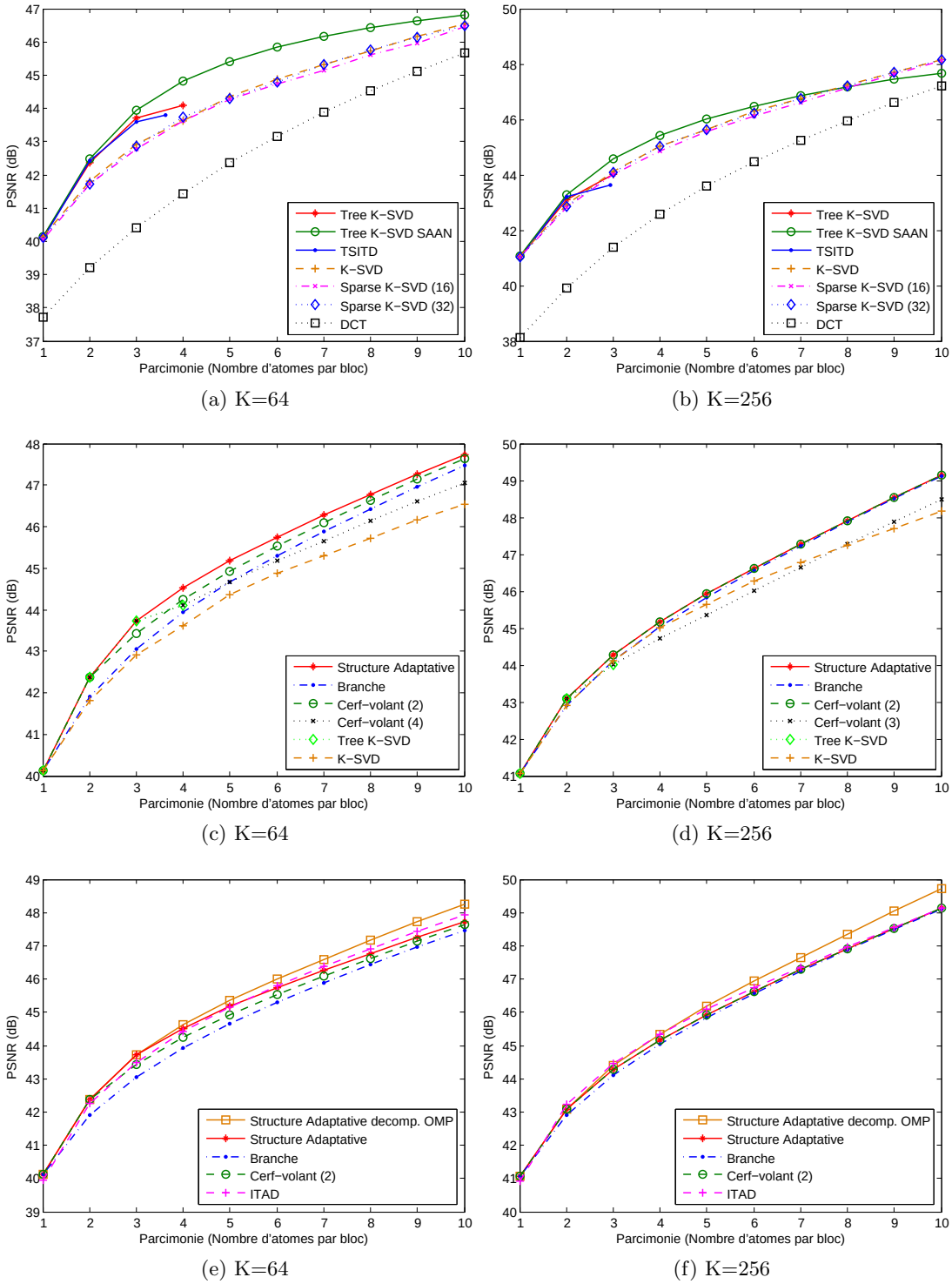



(a) K=64 (b) K=256

(c) K=64 (d) K=256

(e) K=64 (f) K=256

FIGURE 3.21 – Tests PSNR-parcimonie, suite au retrait de la moyenne des patchs, pour K=64 (à gauche) et K=256 (à droite) sur *New York*. Pour des raisons de clarté, les différentes courbes sont affichées sur trois graphiques différents.

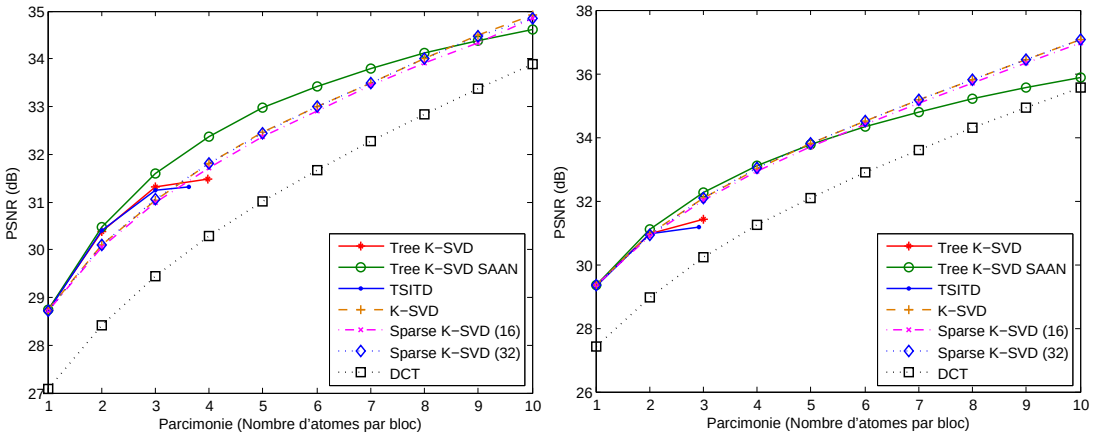



(a) K=64 (b) K=256

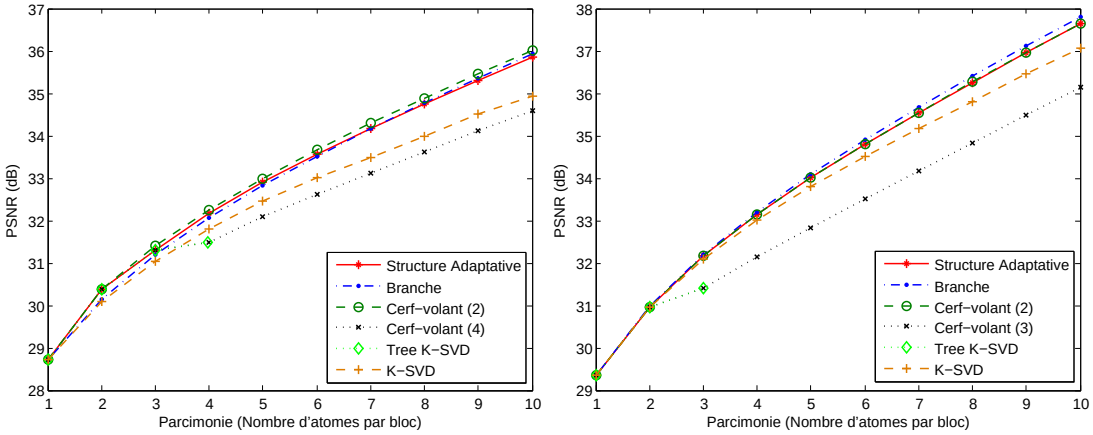
(c) K=64 (d) K=256

(e) K=64 (f) K=256

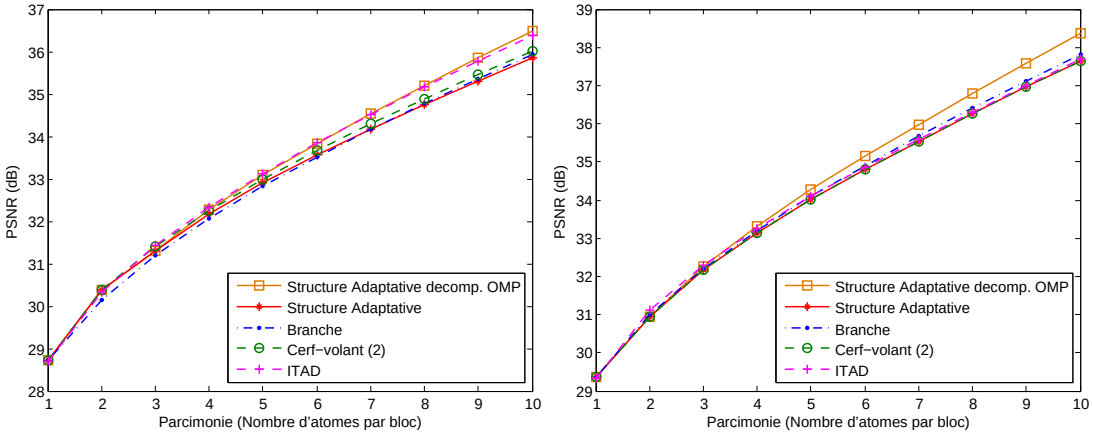
FIGURE 3.22 – Tests PSNR-parcimonie, suite au retrait de la moyenne des patchs, pour K=64 (à gauche) et K=256 (à droite) sur *Désert*. Pour des raisons de clarté, les différentes courbes sont affichées sur trois graphiques différents.




(a) K=64 (b) K=256

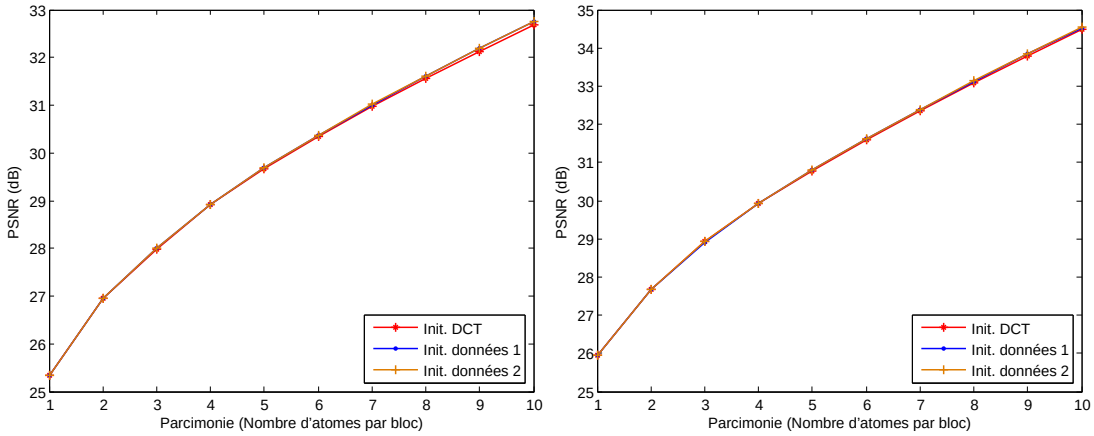



(c) K=64 (d) K=256



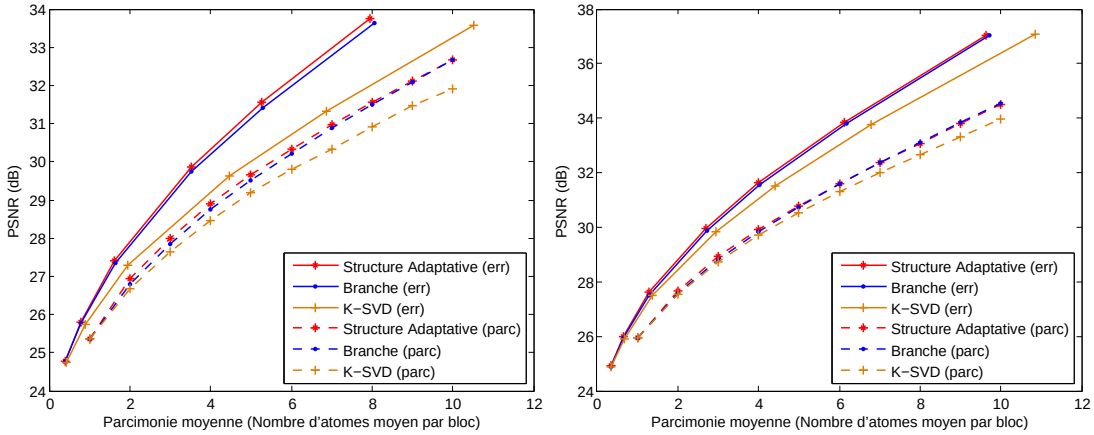

(e) K=64 (f) K=256

FIGURE 3.23 – Tests PSNR-parcimonie, suite au retrait de la moyenne des patchs, pour K=64 (à gauche) et K=256 (à droite) sur *Hambourg*. Pour des raisons de clarté, les différentes courbes sont affichées sur trois graphiques différents.




(a) K=64 (b) K=256

FIGURE 3.24 – Tests d'initialisation de la Structure Adaptative pour K=64 et K=256 sur *New York*.




(a) K=64 (b) K=256

FIGURE 3.25 – Tests PSNR-parcimonie, en utilisant une erreur cible comme critère d'arrêt (err) ou une parcimonie cible (parc), pour K=64 et K=256, sur *New York*.

## 3.6 Conclusion

Dans ce chapitre, nous avons présenté différentes structures de dictionnaires, en partant d'une structure arborescente que nous avons fait évoluer jusqu'à une structure s'adaptant durant l'apprentissage. Ces structures sont davantage adaptées au problème du codage que les dictionnaires "plats" tels que K-SVD et ont pour but d'offrir un meilleur compromis entre erreur de représentation et coût de codage. En effet, ces structures peuvent contenir globalement davantage d'atomes afin d'atteindre une meilleure qualité de représentation, tout en conservant, de part leur structuration en de petits dictionnaires, un coût de codage de chaque indice codé individuellement équivalent et une complexité de décomposition comparable, voire inférieure lorsque OMP est utilisé sur les dictionnaires "plats". De plus, ces structures sont scalables en parcimonie et peuvent donc être utilisées, une fois apprises, pour différentes valeurs de parcimonie, tandis que K-SVD ou Sparse K-SVD sont appris pour une parcimonie donnée.

Nous avons commencé par étudier une structure arborescente nommée Tree K-SVD. Bien qu'efficace sur ses premiers niveaux, la multiplication du nombre de dictionnaires à chaque niveau rend les niveaux plus profonds inefficaces, ce qui pénalise les résultats par la suite. C'est pourquoi la structure en "cerf-volant" propose d'élaguer les différentes branches de l'arbre à un niveau donné pour n'apprendre ensuite qu'un unique dictionnaire par niveau. Cependant, le choix du niveau d'élagage des branches est important et influe sur les résultats différemment selon l'image test. Afin de ne pas imposer un niveau d'élagage commun à toutes les branches, la Structure Adaptative a été créée. Cette structure s'adapte automatiquement durant l'apprentissage d'après les vecteurs d'entraînement. Les branches de l'arbre sont progressivement élaguées selon leur popularité et prolongées par une branche commune. Cette Structure Adaptative présente ainsi une meilleure qualité de reconstruction à parcimonie égale que les dictionnaires "plats" K-SVD ou encore Sparse K-SVD. Son intérêt par rapport à une structure en branche est davantage marqué lorsque la taille des dictionnaires est limitée et tout particulièrement lorsque l'image test se rapproche des images d'apprentissage. Dans le cas contraire, une structure en branche est capable de donner des résultats similaires, voire légèrement supérieurs.

Nous avons de plus pu voir qu'il est intéressant de retirer la moyenne des patchs d'apprentissage, ce qui permet d'équilibrer les structures en équilibrant la popularité des atomes au premier niveau, et ainsi d'améliorer la convergence de K-SVD à ce niveau.

Enfin, nous nous sommes intéressés au critère d'arrêt de la décomposition en montrant le gain obtenu en utilisant une erreur cible plutôt qu'une parcimonie fixe pour représenter chaque bloc. Nous avons ainsi constaté le gain apporté à la qualité de reconstruction par une parcimonie variable entre les blocs de l'image test.

Ces structures de dictionnaires vont, lors du prochain chapitre, être utilisées dans le cadre de la compression d'images satellites. Des codeurs seront développés autour de ces dictionnaires structurés afin de se comparer aux standards de compression d'images.

# Chapitre 4

# Performances des dictionnaires structurés pour le codage

Nous avons dans le chapitre précédent étudié plusieurs structures de dictionnaires, davantage adaptées au problème du codage que des dictionnaires dits “plats” appris par exemple avec l'algorithme K-SVD. En effet, les comparaisons entre les différents dictionnaires ont montré le gain en représentation que les dictionnaires structurés peuvent apporter, à un coût de codage de chaque indice codé individuellement équivalent. Dans ce chapitre, nous allons vérifier les performances de ces structures de dictionnaires du point de vue du codage.

Nous nous situons dans un cas d'étude où le dictionnaire est supposé connu du codeur comme du décodeur et n'est donc pas à transmettre. C'est pourquoi le dictionnaire est appris sur une base variée d'images satellite (la même base que dans le chapitre précédent, voir Annexe 1). Le but est de compresser des images test non comprises dans la base d'apprentissage. On peut imaginer que le dictionnaire est appris et intégré au satellite avant son lancement dans l'espace.

Nous commencerons dans ce chapitre par comparer les performances de codage des différents dictionnaires en appliquant un schéma de codage simple. Puis nous étudierons un schéma de codage, similaire à celui d'ITAD [ZGK11], optimisé autour de la Structure Adaptative. Les performances de ce codeur seront alors comparées à des codeurs de l'état de l'art. Ensuite, nous tenterons d'intégrer des dictionnaires structurés appris au sein de HEVC Intra [SOHW12, LBH$^+$12], lors de l'étape de transformation, afin d'étudier leur intérêt dans HEVC vis-à-vis de la transformation DCT usuelle.

Enfin, nous spécialiserons les dictionnaires à une scène particulière dans le cadre de l'observation persistante, en travaillant sur des séquences d'images de scènes quasiment statiques. L'apprentissage et les tests seront alors réalisés sur les images d'une même séquence afin d'étudier les performances des dictionnaires structurés dans un cas quasiment idéal, où les données de test et d'apprentissage sont très proches, plus favorable que le cas d'étude traité précédemment.

## 4.1 Performances de codage

Utiliser les représentations parcimonieuses dans un contexte de codage revient à coder et transmettre pour chaque bloc de l'image test un certain nombre, selon la parcimonie appliquée pour chaque bloc, de paires coefficient/indice issues de la décomposition sur les dictionnaires. De plus, la valeur moyenne de chaque bloc étant retirée en amont de la décomposition, ces valeurs moyennes doivent également être codées.

Dans cette section, ces trois entités vont être codées pour chaque bloc de manière relativement simple dans le but de comparer les différentes structures de dictionnaires dans le cadre de la compression.

### 4.1.1 Description du codeur initial non optimisé

Les coefficients sont tout d'abord quantifiés par une quantification scalaire uniforme à zone morte. La zone morte permet de mettre à zéro des coefficients trop faibles et donc peu significatifs. La quantification suit ainsi la formule suivante :

$$\hat{x} = round\,(\frac{x}{Q}) \tag{4.1}$$

avec $\hat{x}$ l'index de quantification du coefficient $x$, $Q$ le pas de quantification et *round* la fonction d'arrondi à l'entier le plus proche. Le coefficient quantifié $\tilde{x}$, reconstruit après quantification inverse, est simplement obtenu par une multiplication de l'index de quantification par le pas de quantification :

$$\tilde{x} = \hat{x} \times Q \tag{4.2}$$

De cette façon, tous les coefficients dans la zone morte définie par l'intervalle $]-\frac{Q}{2};\frac{Q}{2}[$ sont mis à zéro par la quantification, et ne seront pas codés.

Les indices de quantification $\hat{x}$ non nuls sont ensuite placés à la suite sous la forme d'une séquence. Pour chaque bloc, ils sont placés dans l'ordre des niveaux (dans la structure) des atomes auxquels ils correspondent pour les dictionnaires structurés, et dans l'ordre des indices des atomes correspondant pour les dictionnaires "plats", sans '0' intermédiaires dans la séquence. Chaque bloc est terminé par un code "EOB" (*End Of Block*). Cette séquence est alors codée pas un codage entropique de Huffman similaire à celui réalisé au sein de JPEG pour les coefficiens AC. La même table usuelle est utilisée, à la différence près que seule la première partie de la table correspondant à un *run* de zéro '0' est utile, étant donné que la séquence ne contient aucune valeur nulle. Ainsi, chaque indice de quantification de coefficient $\hat{x}$ est codé via deux symboles : le premier est un code de Huffman, contenant les informations *runlength* (nombre de 0 précédents) et *size* (taille du second symbole), et le second est un code à longueur variable contenant l'information d'amplitude (voir section 2.1.1.3).

Les indices des atomes sont quant à eux représentés par un code à longueur fixe de $R$ bits pour chaque indice avec :

$$R = log_2(K) \tag{4.3}$$

où $K$ correspond au nombre d'atomes de chaque dictionnaire au sein des dictionnaires structurés, ou au nombre d'atomes de l'unique dictionnaire pour les dictionnaires "plats".

Enfin, chaque valeur de moyenne de bloc est arrondie à l'entier le plus proche et codée sur 8 bits, cette valeur étant comprise pour une image 8 bits entre 0 et 255.

### 4.1.2 Comparaison des différentes structures

La décomposition et le codage sont réalisés séparément. La décomposition est d'abord effectuée sur les différentes structures de dictionnaires pour une parcimonie de 10 atomes, ce qui représente le nombre maximal d'atomes pouvant être codés pour chaque bloc. Les structures apprises sur 10 niveaux pour le chapitre 3, ainsi que les dictionnaires K-SVD appris pour une parcimonie de 10 atomes, peuvent de cette façon être utilisés. Puis les coefficients sont quantifiés et codés, ainsi que les indices et les valeurs moyennes des blocs.

Les courbes débit-distorsion sont obtenues en faisant varier le pas de quantification $Q$. En plus de permettre une quantification plus précise, diminuer le pas permet de diminuer la taille de la zone morte afin de quantifier moins de coefficients à zéro et donc de coder davantage de coefficients (avec une limite fixée à 10) pour atteindre une meilleure qualité de représentation. Le pas de quantification permet ainsi de réguler le nombre d'atomes codés pour chaque bloc. En effet, pour les dictionnaires "plats", seuls les coefficients non quantifiés à zéro sont codés. Pour les dictionnaires structurés, le codage des coefficients d'un bloc, traités dans l'ordre niveau après niveau en partant du premier, est arrêté dès qu'un coefficient est mis à zéro par la quantification afin de ne pas coder de coefficient nul, en considérant que les coefficients suivants sont peu significatifs. En effet, les coefficients sont en général de moins en moins importants. Il est cependant possible qu'un (voire plusieurs) des coefficients suivants ne soit pas mis à zéro par la quantification. Ce dernier n'est alors pas codé puisqu'on ne souhaite pas coder de coefficient nul, et surtout l'indice associé, pour pouvoir coder ce coefficient suivant non nul.

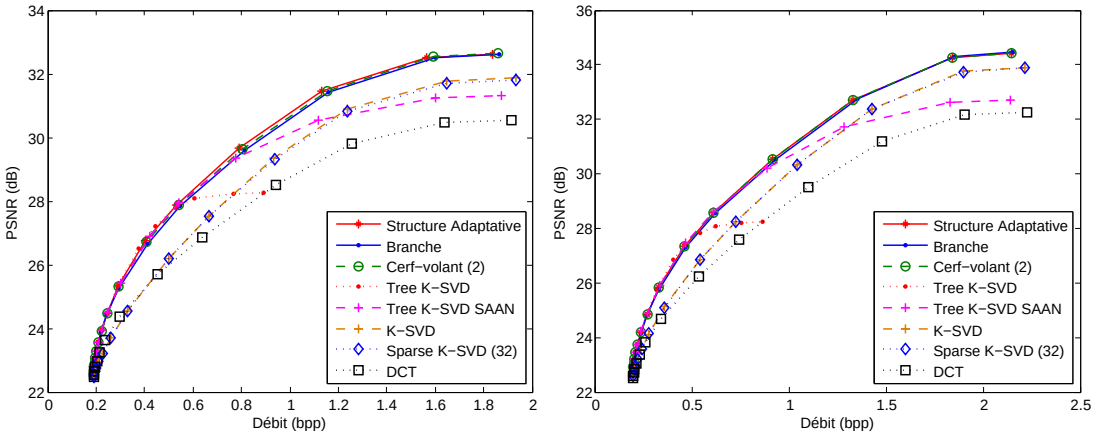
Comme pour les tests en fonction de la parcimonie, réalisés lors du chapitre précédent, un premier constat permet d'apprécier le gain apporté par des dictionnaires appris vis-à-vis du dictionnaire DCT prédéfini (Fig. 4.1-4.3). De plus, K-SVD et Sparse K-SVD (avec une parcimonie de 32 valeurs non nulles sur les atomes du dictionnaire appris) permettent d'obtenir des résultats de codage quasiment identiques.

Tree K-SVD, la structure arborescente, offre des résultats supérieurs à ceux de K-SVD à bas débit, lorsque les atomes codés sont sélectionnés dans les premiers niveaux de l'arbre (Fig. 4.1-4.3). Puis la structure perd en efficacité lorsque le débit augmente et que les atomes sont sélectionnés à des niveaux plus profonds de l'arbre. Le PSNR stagne ensuite à cause de l'arrêt des branches. La SAAN appliquée à Tree KSVD permet de prolonger l'efficacité de l'arbre, au prix d'une complexité de décomposition supérieure. A noter qu'une pénalité de 1 bit par atome (sauf pour le dernier atome de chaque bloc) est appliquée pour Tree K-SVD SAAN au codage, ce bit correspondant à un *flag* signalant si le prochain atome est sélectionné au niveau courant ou au niveau suivant.

Mais la courbe du Tree K-SVD SAAN est finalement rattrapée par celle du K-SVD lorsque le débit augmente, hormis sur l'image *Désert*. Cette image étant plus proche des images d'apprentissage, elle profite davantage des structures développées.

La Structure Adaptative et la structure en "cerf-volant" refermée après 2 niveaux présentent des résultats globalement proches (Fig. 4.1-4.3).

Enfin, les résultats de la Structure Adaptative et de la structure en branche sont similaires pour K=256 (Fig. 4.1-4.3 (b)). La Structure Adaptative est cependant légèrement plus performante pour de petits dictionnaires (K=64), lorsque l'image test est proche des images d'apprentissages, comme c'est le cas pour l'image *Désert* (Fig. 4.2 (a)), et à moindre mesure pour *New York* (Fig. 4.1 (a)).

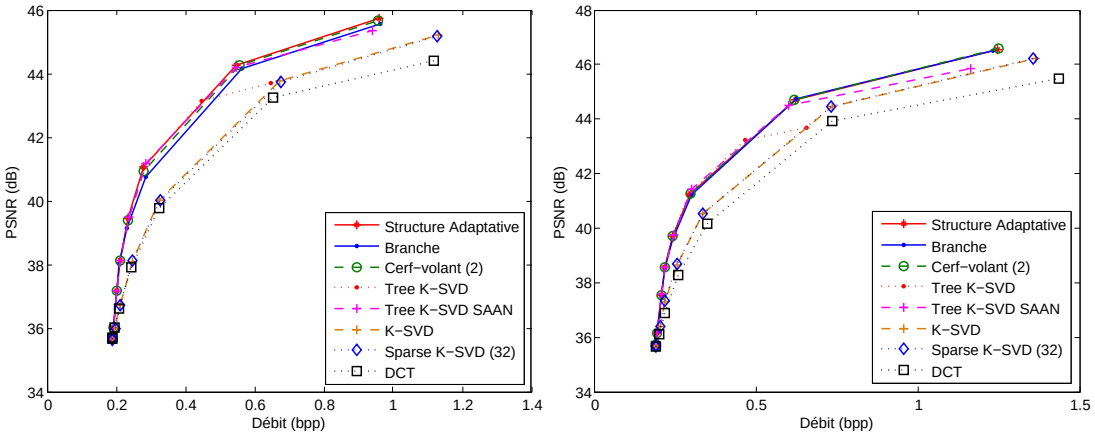



(a) K=64 (b) K=256

FIGURE 4.1 – Performances de codage des différentes structures de dictionnaires pour K=64 et K=256 sur *New York*.

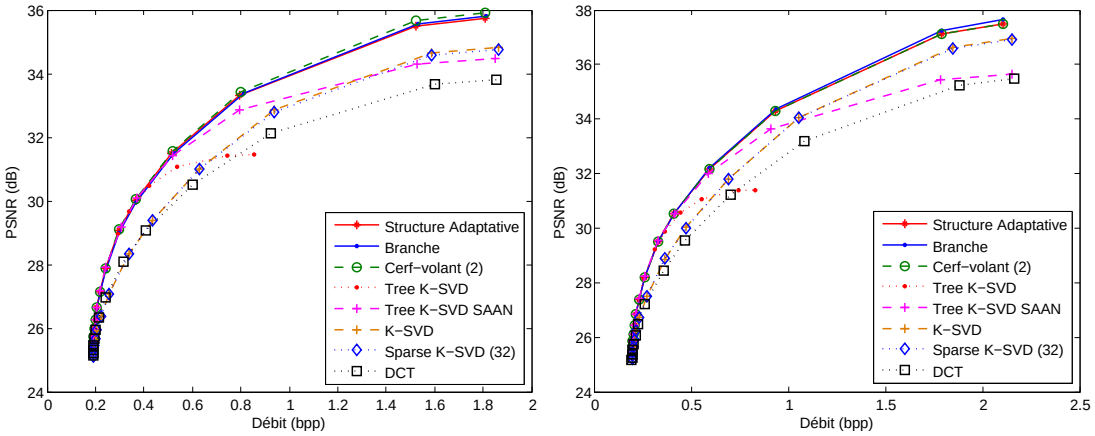
La régulation du nombre de coefficients codés par bloc par le pas de quantification $Q$ permet de coder davantage de coefficients pour les blocs présentant des textures complexes que pour les blocs homogènes (Fig. 4.4), dont les coefficients sont très vite faibles et donc mis à zéro par la quantification. Seuls les coefficients les plus significatifs sont ainsi codés. On remarque que le bas de l'image est plus sombre car il nécessite globalement moins d'atomes pour la représentation du fait que cette partie de l'image test (bas de l'image test Fig. 3.3) est très proche d'une partie d'une image d'apprentissage (haut de l'image d'apprentissage Annexe 1 Fig. 22(l)), cette zone étant commune entre les deux images, et avec le même quadrillage des blocs.

Nous souhaitons désormais comparer, pour ce schéma de codage, les structures de dictionnaires pour différentes tailles de dictionnaires $K$ sur le même graphique (Fig. 4.5). Pour K-SVD, le dictionnaire de 256 atomes présente un gain par rapport au dictionnaire de 64 atomes. Le surplus de codage des indices dû à un dictionnaire plus large est donc compensé par le gain en qualité apporté par un plus grand choix d'atomes et par l'obtention de représentations plus parcimonieuses. Pour la structure en branche et plus particulièrement pour la Structure Adaptative, le choix de $K$ ne semble pas si




(a) K=64 (b) K=256

FIGURE 4.2 – Performances de codage des différentes structures de dictionnaires pour K=64 et K=256 sur *Désert*.




(a) K=64 (b) K=256

FIGURE 4.3 – Performances de codage des différentes structures de dictionnaires pour K=64 et K=256 sur *Hambourg*.

prépondérant jusqu'à environ 1 bpp. Ces structures semblent donc moins sensibles au choix de ce paramètre dans le cadre de ce schéma de codage. En augmentant le débit, les dictionnaires de 256 atomes prennent forcément le dessus sur ceux de 64 atomes de part la limite de parcimonie fixée à 10 atomes. A parcimonie de 10 atomes, un dictionnaire plus large permet en effet d'atteindre une meilleure qualité de reconstruction, l'asymptote des courbes pour $K = 256$ est donc logiquement plus haute. En s'autorisant une limite de parcimonie supérieure à 10 atomes, la divergence entre les courbes pour $K = 64$ et $K = 256$ se produirait donc à un débit plus grand.

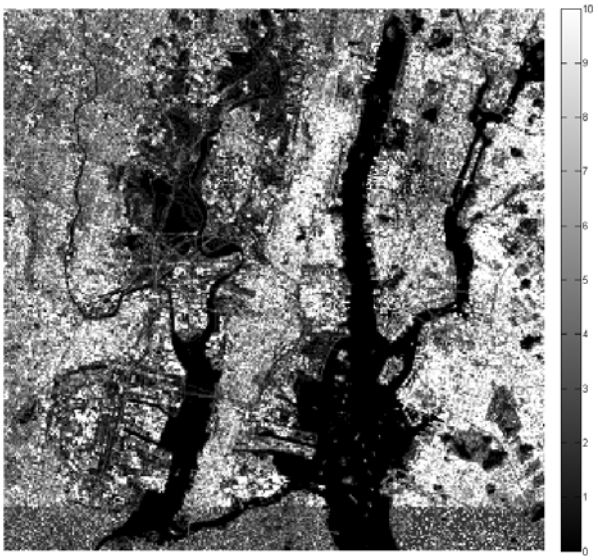



FIGURE 4.4 – Carte de la distribution du nombre d'atomes sélectionnés par bloc sur *New York* pour la Structure Adaptative (K=256) (point correspondant à un PSNR de 30.53 dB et un débit de 0.91 bpp).

JPEG 2000 est ensuite ajouté à la comparaison (Fig. 4.5). Le codage par JPEG 2000 est réalisé à l'aide du logiciel OPEN JPEG [UCL], dans sa version 2.0.0. Cette librairie est développée et maintenue par l'Université Catholique de Louvain (UCL). Elle est appliquée avec les paramètres par défaut, c'est-à-dire que toute l'image est considérée comme une seule tuile et un *precinct* (voir le chapitre 2 pour la définition des différents termes), chaque *code-block* est de taille $64 \times 64$, et 5 décompositions en ondelettes sont réalisées (6 niveaux de résolution). Cependant, une compression irréversible est appliquée avec le filtre 9/7 qui est le plus efficace. La courbe débit-distorsion est obtenue en faisant varier le taux de compression. On remarque alors que les résultats de JPEG 2000, un codeur optimisé, sont nettement supérieurs à ceux obtenus avec notre codeur. Très simple dans son fonctionnement, ce dernier nous a permis de comparer les différentes structures de dictionnaires dans un contexte de codage. Nous allons désormais chercher à optimiser ce codeur pour les dictionnaires structurés afin d'en améliorer les performances dans le but de se rapprocher de JPEG 2000.

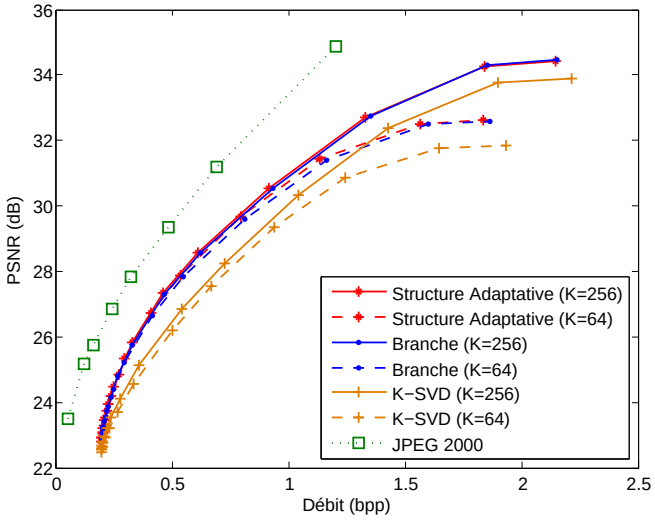



FIGURE 4.5 – Performances de codage pour différentes tailles de dictionnaires (K) sur *New York* et comparaison avec JPEG 2000.

## 4.2 Un codeur optimisé pour les dictionnaires structurés

Les résultats obtenus avec le codeur simple, présentés précédemment et inférieurs à ceux de JPEG 2000, nous incitent à davantage optimiser le codeur, en particulier pour les dictionnaires structurés. Nous utilisons dans cette optique un codeur similaire à celui utilisé pour ITAD [ZGK11, Zep10], adapté à la Structure Adaptative et la structure en branche (Fig. 4.6).

Le codeur fonctionne pour un débit cible, comme c'est le cas pour le codeur CCSDS [fSDS05], et cherche à optimiser la qualité de reconstruction d'une image pour ce débit (Fig. 4.6). Le pas de quantification $Q$ des coefficients, et celui $\Delta_{DC}$ des valeurs moyennes, sont également des paramètres du codeur. Contrairement au codeur simple précédent, la décomposition et le codage sont désormais liés et réalisés de façon progressive.

Les structures utilisées par ce codeur, la Structure Adaptative ou la structure en branche, sont apprises pour K=256 (taille de chaque dictionnaire au sein des structures) et sur 20 niveaux, de telle sorte que la parcimonie maximale pour toute décomposition de bloc soit fixée à 20 atomes plutôt que 10 comme précédemment. Le fait d'augmenter le nombre maximal d'atomes pouvant être sélectionnés par bloc lors de la décomposition permet de moins limiter les résultats à plus haut débit, puisque cela augmente la qualité de reconstruction maximale atteignable pour chaque bloc.

### 4.2.1 Description du codeur optimisé

#### Codage des valeurs moyennes

Les valeurs moyennes de chaque bloc sont tout d'abord codées. Comme cela est

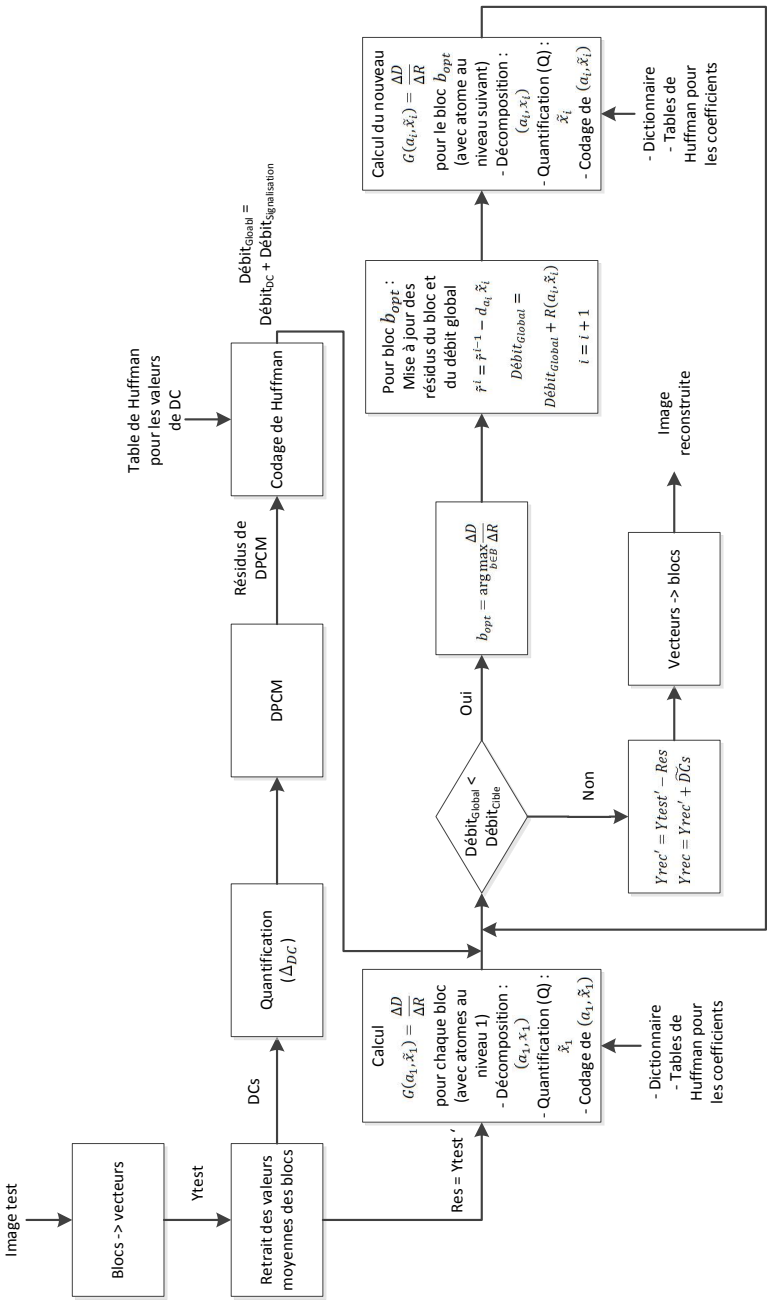



FIGURE 4.6 – Schéma du codeur optimisé pour les dictionnaires structurés.

réalisé dans [XLLZ14], chaque valeur moyenne DC est quantifiée en la divisant par une valeur constante $\Delta_{DC}$, suivie d'un arrondi à l'entier le plus proche : $round(\frac{DC}{\Delta_{DC}})$. Comme dans cet article, la valeur de $\Delta_{DC}$ est choisie à 4. Une valeur constante de $\Delta_{DC}$ est choisie, quelque soit le débit, pour des raisons de simplicité, mais l'utilisation d'un $\Delta_{DC}$ variable pourrait améliorer le codage des valeurs moyennes. Ensuite, une prédiction DPCM sans perte est appliquée à ces valeurs de moyennes quantifiées. La DPCM ici appliquée calcule simplement une différence par rapport à un bloc voisin. Chaque valeur moyenne de bloc est ainsi prédite d'après la valeur de moyenne du bloc au-dessus, hormis pour les blocs correspondant aux premières lignes de l'image, prédits par rapport au bloc voisin de gauche (Fig. 4.7). Enfin, un codage de Huffman est appliqué afin d'encoder les résidus de la prédiction DPCM, avec une table de Huffman apprise sur les résidus de DPCM appliquées aux valeurs moyennes quantifiées des images d'apprentissage.

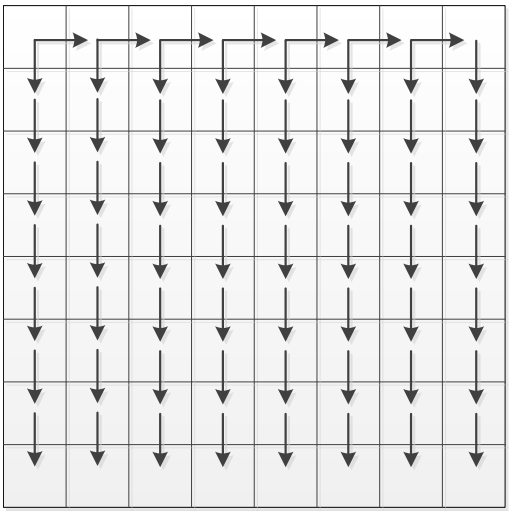


FIGURE 4.7 – Prédictions pour la DPCM.

**Codage des coefficients**

Les coefficients sont comme précédemment quantifiés par une quantification scalaire uniforme à zone morte. Puis un codage de Huffman est appliqué, en apprenant grâce aux données d'apprentissage, pour chaque pas de quantification Q différent, une table de Huffman par niveau de la structure. Chaque table est apprise sur les coefficients de décomposition quantifiés des données d'apprentissage à un niveau de la structure donné. Ainsi, un code de Huffman est associé à chaque niveau de reconstruction des valeurs de coefficients. Contrairement au schéma ITAD, les tables sont seulement apprises sur des coefficients non mis à zéro par la quantification, de façon à ne pas attribuer un code très court au niveau de reconstruction 0, étant donné qu'aucun coefficient (des vecteurs de test) mis à zéro par la quantification ne sera codé. La décomposition de chaque vecteur

d'apprentissage est ainsi arrêtée dès qu'un coefficient est quantifié à zéro.

### Codage des indices

Un code à longueur fixe est utilisé pour coder chaque indice, de telle sorte que le coût associé à un indice sélectionné dans un dictionnaire de $K$ atomes est : $R(a_i) = log_2(K)$. En calculant l'entropie des indices des atomes sélectionnés sur la Structure Adaptative, on obtient une valeur proche du coût du code à longueur fixe, la distribution des indices étant relativement uniforme. D'où ce choix de codage pour les indices. Observons par exemple au point 0.5 bpp la distribution des indices des atomes choisis au premier niveau de la Structure Adaptative pour l'image test *New York* (Fig. 4.8). L'entropie de ces indices est alors de 7.94 bits environ, très proche des 8 bits du code à longueur fixe. La même stratégie est employée dans [ZGK11] ou encore [XLLZ14] pour le codage des indices.

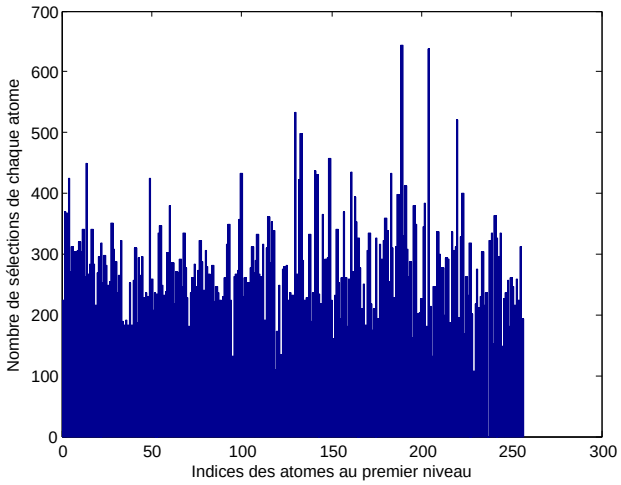



FIGURE 4.8 – Distribution des indices des atomes sélectionnés au premier niveau de la Structure Adaptative à 0.5 bpp sur *New York*.

### Schéma du codeur optimisé

Le schéma de codage utilisé (Fig. 4.6) est similaire à celui décrit dans [ZGK11] pour la structure de dictionnaires ITAD : le codage des valeurs moyennes des blocs est d'abord réalisé, puis à chaque étape, un atome est ajouté à la décomposition d'un bloc afin d'en améliorer la représentation, jusqu'à atteindre un débit cible. L'objectif est ainsi d'atteindre la meilleure représentation possible de l'image, en représentant chaque bloc avec une parcimonie variable, pour un certain débit cible. Le même critère débit-distorsion est utilisé afin de choisir à chaque étape le bloc de l'image auquel ajouter un nouvel atome, le bloc choisi étant celui maximisant le rapport entre la diminution de la distorsion sur le coût de codage liés à l'ajout de ce nouvel atome dans la représentation.

Ce gain $G$ est ainsi exprimé :

$$\begin{aligned} G(a_i, \tilde{x_i}) = \frac{\Delta D}{\Delta R} &= \frac{||y - \tilde{y}^{i-1}||_2^2 - ||y - \tilde{y}^{i}||_2^2}{R(a_i, \tilde{x_i})} \\ &= \frac{||\tilde{r}^{i-1}||_2^2 - ||\tilde{r}^{i}||_2^2}{R(a_i, \tilde{x_i})} = \frac{||\tilde{r}^{i-1}||_2^2 - ||\tilde{r}^{i-1} - d_{a_i}\tilde{x_i}||_2^2}{R(a_i, \tilde{x_i})} \end{aligned} \tag{4.4}$$

avec $d_{a_i}$ le nouvel atome, à l'indice $a_i$ d'un dictionnaire au niveau $i$, et $\tilde{x_i}$ le coefficient associé reconstruit (après quantification et quantification inverse). $\tilde{y}^i$ représente l'approximation du bloc $y$ au niveau $i$, c'est-à-dire avec $i$ atomes sélectionnés pour la représentation, en prenant en compte la quantification des coefficients, $\tilde{r}^i$ représentant les résidus de l'approximation $\tilde{y}^i$. Le débit associé à ce nouvel atome, $R(a_i, \tilde{x_i})$, correspond au coût de codage du coefficient, c'est-à-dire à la longueur du code de Huffman associé (dans la table de Huffman de niveau $i$, apprise grâce aux images d'apprentissage pour le pas de quantification $Q$ choisi initialement), au coût de l'indice $R(a_i)$ et à un coût de signalisation "EOB" de 1 bit signalant si cet atome est le dernier atome du bloc à coder ou non (bit à '1' dans un cas et à '0' dans l'autre).

Enfin, contrairement au codeur ITAD où cela n'est pas nécessaire [Zep10], la quantification des coefficients est ici intégrée dans la décomposition. En effet, chaque vecteur de résidus est mis à jour d'après le coefficient reconstruit après quantification et quantification inverse, et non le coefficient réel calculé par l'algorithme MP. On cherche ainsi à compenser les erreurs de quantification apportées à chaque niveau en les prenant en compte pour la sélection des atomes et le calcul des coefficients aux niveaux suivants.

### 4.2.2 Performances débit-distorsion du codeur optimisé

#### Choix des paramètres du codeur

Le codeur optimisé dépend de trois paramètres : le pas de quantification des valeurs moyennes $\Delta_{DC}$, le pas de quantification des coefficients $Q$ et le débit cible.

En traçant pour différentes valeurs du pas de quantification des coefficients $Q$ la courbe débit-distorsion (Fig. 4.9), on remarque que le pas optimal dépend du débit. Ainsi, un pas important est préférable à bas débit, de façon à avoir des tables de Huffman de taille réduite. Puis lorsque le débit augmente, le pas optimal diminue, de telle sorte que la quantification des coefficients s'affine, les tables de Huffman contenant davantage de codes et donc davantage de niveaux de reconstruction. Cette observation est cohérente avec les observations faites dans [ZGK11], même si un pas de quantification constant quelque soit le débit y est finalement choisi, ou encore [XLLZ14] où la taille des tables de codes pour la quantification augmente avec le débit. L'enveloppe de ces courbes obtenues pour différentes valeurs de $Q$ nous donne la courbe débit-distorsion optimale pour l'image test. En pratique, pour chaque image test, on cherche en chaque point de débit le pas de quantification optimal donnant la meilleure qualité de reconstruction.

Idéalement, le pas de quantification des valeurs de moyenne $\Delta_{DC}$ devrait être également progressivement diminué. Cependant, son impact se limitant au codage des

valeurs de moyenne et pour des raisons de simplicité, une valeur constante de $\Delta_{DC}$ est appliquée pour chaque image test quelque soit le débit. Cette valeur est fixée à 4, comme cela est réalisé dans [XLLZ14], ce qui améliore significativement les résultats à bas débit par rapport à une valeur de 1. Lorsque le débit global augmente, la proportion du débit alloué au codage des valeurs moyenne devient moins significative et le gain diminue progressivement. Cependant, pour l'image *Désert*, la valeur du $\Delta_{DC}$ est fixée à 1 du fait de l'homogénéité de l'image.

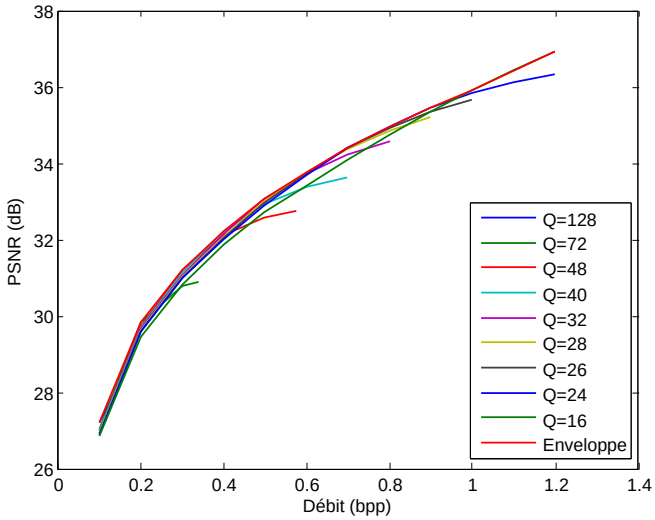



FIGURE 4.9 – Courbes débit-distorsion pour différents pas de quantification Q, obtenues avec la Structure Adaptative et le codeur optimisé, sur *Hambourg* ($\Delta_{DC} = 4$).

**Performances débit-distorsion de la Structure Adaptative**

Les performances en débit-distorsion de la Structure Adaptative associée au codeur optimisé sont comparées avec le codec ITAD, ainsi que les codeurs d'images état de l'art JPEG 2000, en utilisant l'ondelette 9/7 ou 5/3 (toujours avec OPEN JPEG [UCL]), CCSDS 122 et JPEG (Fig. 4.11 - 4.14).

La structure ITAD est apprise, comme la Structure Adaptative, sur 20 niveaux, chaque dictionnaire au sein de la structure comportant 256 atomes également. Afin de coder les valeurs moyennes au sein du codec ITAD, celles-ci sont d'abord arrondies à l'entier le plus proche puis codées sur 8 bits chacune, les valeurs étant comprises entre 0 et 255. Comme pour la Structure Adaptative, le pas de quantification optimal est recherché pour chaque point de débit cible et diminue ainsi avec le débit.

Afin de tester le codeur CCSDS, nous utilisons une implémentation du standard CCSDS 122.0-B-1 [CCS] de l'université de Nebraska [WSB]. Les images tests sont codées et décodées pour différents débits cibles, en utilisant l'ondelette 9/7 réelle et la valeur par défaut de 256 blocs par segment.

Pour JPEG, le logiciel Vc Demo [TD], dans sa version 5.03, est utilisé. Ce logiciel a été développé par le groupe *Information and Communication Theory* (ICT) de l'Uni-

versité Technologique de Delft (TU-Delft). Les images sont encodées pour différents débits avec les paramètres par défaut.

Les trois mêmes images test que précédemment sont utilisées : *New York, Désert et Hambourg*, ainsi que l'image *Los Angeles* (Fig. 4.10). La base d'apprentissage ne possédant pas d'image de Los Angeles, cette image est comme *Hambourg* davantage éloignée de la base que *Désert* et *New York*.

FIGURE 4.10 – Image test : *Los Angeles* (2400x2400) (FTM de 0.36).

En observant les résultats pour l'image test *New York* (Fig. 4.11), assez proche des images d'apprentissage, on note tout d'abord que la Structure Adaptative surpasse ITAD, grâce aux améliorations apportées à l'apprentissage des tables de Huffman pour le codage des coefficients et au codage des valeurs moyennes des blocs, ainsi qu'à l'intégration de la quantification des coefficients dans la décomposition. On peut ensuite voir que JPEG donne des résultats clairement inférieurs aux autres codeurs testés. CCSDS, utilisant la même ondelette 9/7 que JPEG 2000 (9/7), mais réalisant des compromis entre qualité et complexité, présente finalement des résultats similaires à ceux de JPEG 2000 (5/3) utilisant une transformée en ondelette moins complexe mais moins efficace, hormis à très bas débit où ils sont inférieurs. La Structure Adaptative permet d'obtenir des résultats équivalents voire supérieurs à JPEG 2000 (5/3) et CCSDS entre 0.2 et 0.9 bpp, et parvient même à atteindre ceux de JPEG 2000 (9/7) entre 0.3 et 0.5 bpp, malgré un codage entropique simple comparé à JPEG 2000. A plus haut débit (au-dessus de 0.9 bpp), la Structure Adaptative perd ensuite en efficacité par rapport aux standards de compression.

Pour les images *Hambourg* (Fig. 4.12) et *Los Angeles* (Fig. 4.13), davantage éloignées des images d'apprentissage, la Structure Adaptative parvient à atteindre, voire dépasser, les résultats de JPEG 2000 (5/3) et CCSDS sur une certaine plage de débit, avant d'être dépassée à plus haut débit. De plus, ses résultats se rapprochent de ceux de JPEG 2000 (9/7) entre 0.2 et 0.4 bpp.

Enfin, pour l'image *Désert* (Fig. 4.14), particulière car très homogène, la Structure Adaptative est globalement au-dessus de JPEG et JPEG 2000 (5/3), mais est vite dépassée par CCSDS. L'apprentissage est probablement moins utile sur une image de ce genre ne présentant que peu de structures particulières pouvant être apprises pour améliorer leur représentation. JPEG 2000 (9/7) est encore le codeur le plus efficace parmi les codeurs comparés.

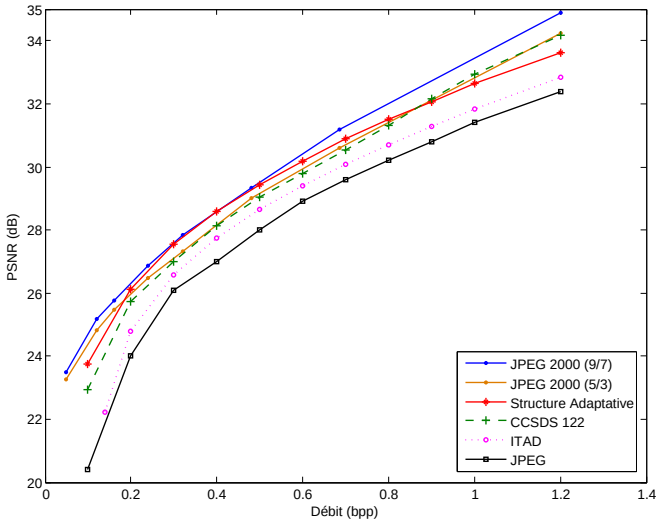



FIGURE 4.11 – Courbes débit-distorsion sur *New York* ($\Delta_{DC} = 4$).

A noter que le codeur optimisé peut également être utilisé avec la structure en branche ($K = 256$, 20 niveaux) et donne pour ces images des résultats similaires à ceux obtenus avec la Structure Adaptative.

### Carte de la distribution du nombre d'atomes sélectionnés par bloc

L'optimisation débit-distorsion permet de sélectionner les blocs auxquels ajouter des atomes dans la représentation jusqu'à atteindre un débit cible. Cela permet de réguler la parcimonie par bloc selon la difficulté de représentation de chaque bloc (Fig. 4.15). En effet, davantage d'atomes sont accordés à la représentation des blocs les plus complexes contenant des textures particulières. Tandis que de nombreux blocs homogènes ne sont représentés que par leur valeur moyenne, c'est-à-dire qu'aucun atome n'est utilisé dans leur représentation, et ce même en augmentant le débit. L'optimisation débit-distorsion semble donc efficace pour attribuer le débit disponible aux blocs le nécessitant afin d'optimiser la qualité de représentation globale de l'image.

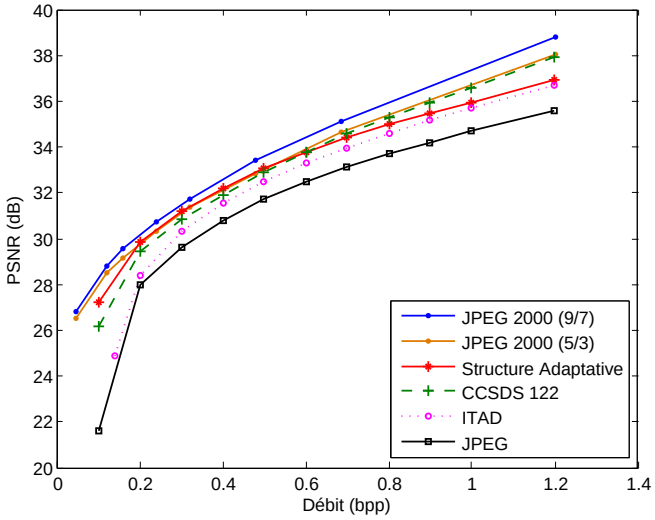



FIGURE 4.12 – Courbes débit-distorsion sur *Hambourg* ($\Delta_{DC} = 4$).

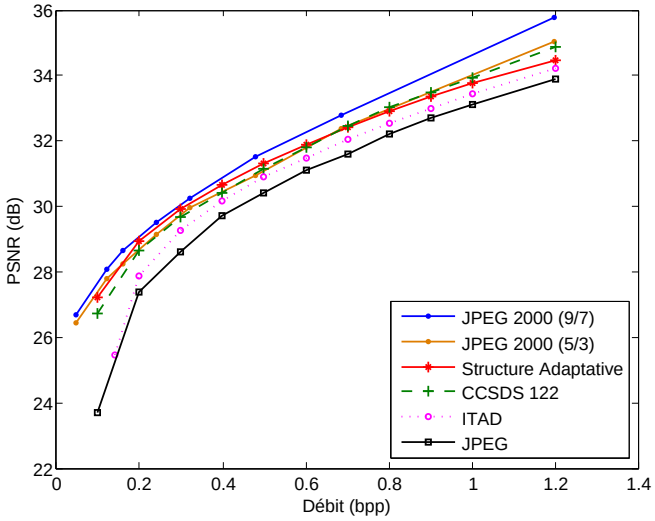



FIGURE 4.13 – Courbes débit-distorsion sur *Los Angeles* ($\Delta_{DC} = 4$).

### Qualité des images reconstruites

Observons maintenant les images reconstruites suite au décodage, pour *New York* (Fig. 4.16 - 4.19) et *Hambourg* (Fig. 4.20 - 4.23), obtenues par la Structure Adaptative et le codeur associé, et par les standards JPEG 2000 (avec l'ondelette 9/7) et CCSDS, utilisant une transformée en ondelettes. Les images sont obtenues pour un débit de 0.7 bpp et seules des portions des images test reconstruites sont montrées. La première portion (a) est de taille 500x500 pixels et la seconde (b), prise sur la première portion, est de taille 150x150 pixels et affichée à l'échelle 1 :1.

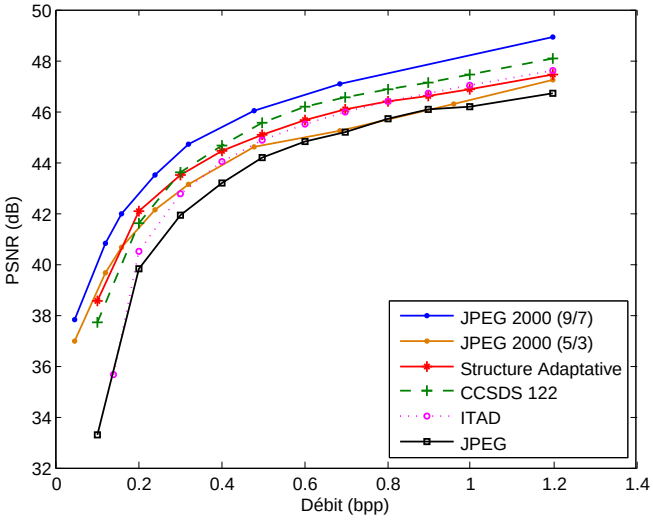
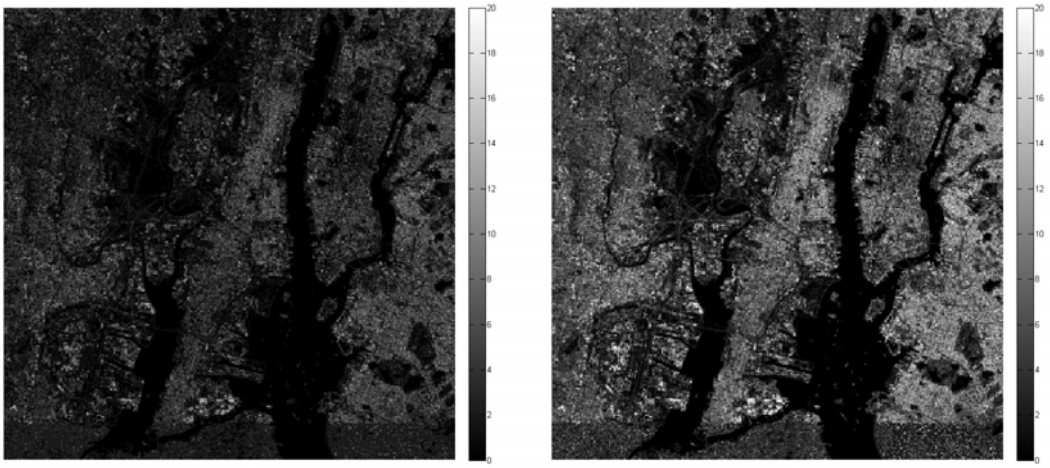



FIGURE 4.14 – Courbes débit-distorsion sur *Désert* ($\Delta_{DC} = 1$).



(a) Débit de 0.5 bpp (b) Débit de 0.9 bpp

FIGURE 4.15 – Carte de la distribution du nombre d'atomes sélectionnés par bloc sur *New York* avec la Structure Adaptative (K=256) pour deux débits cibles : 0.5 bpp (a) et 0.9 bpp (b).

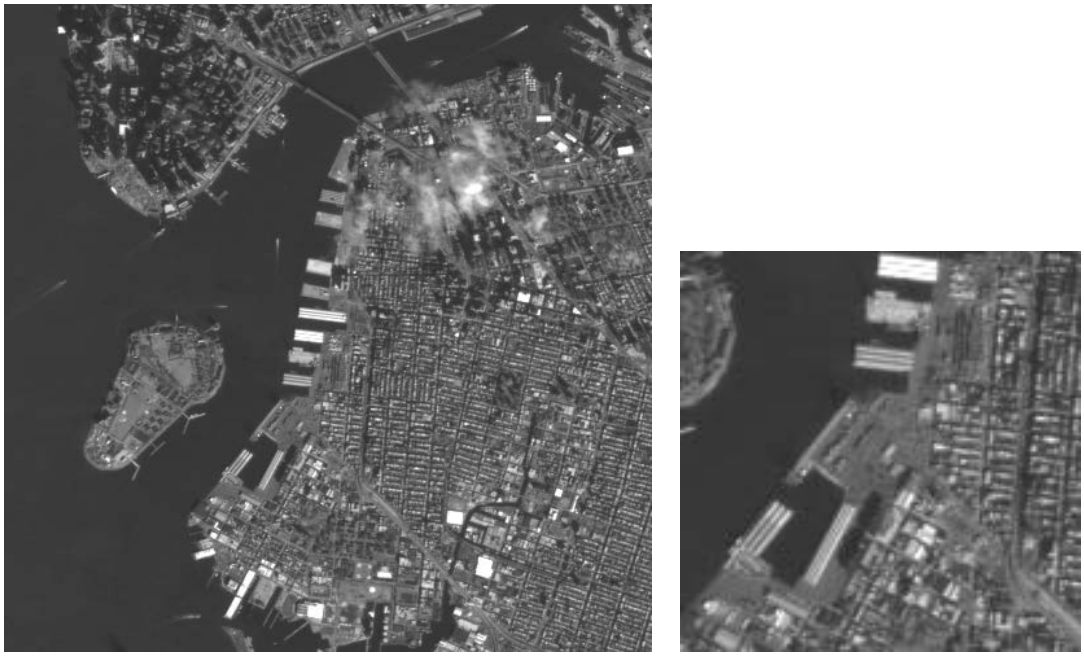
La première observation est que les zones structurées et avec des contrastes importants sont bien représentées par la Structure Adaptative grâce à l'apprentissage et au schéma de codage leur accordant davantage d'atomes que sur les zones homogènes ou peu contrastées, où de nombreux blocs ne sont représentés que par leur valeur moyenne.

Par rapport à JPEG 2000 et CCSDS, utilisant des transformées en ondelettes, la Structure Adaptative permet de reconstruire des images moins floues. Les bords sont

plus nets, comme on peut le voir par exemple sur les ponts ou les côtes, et certains détails sont davantage préservés (voir par exemple sur les vignettes 150x150 de *New York* ou sur la partie supérieure gauche de l'image 500x500).

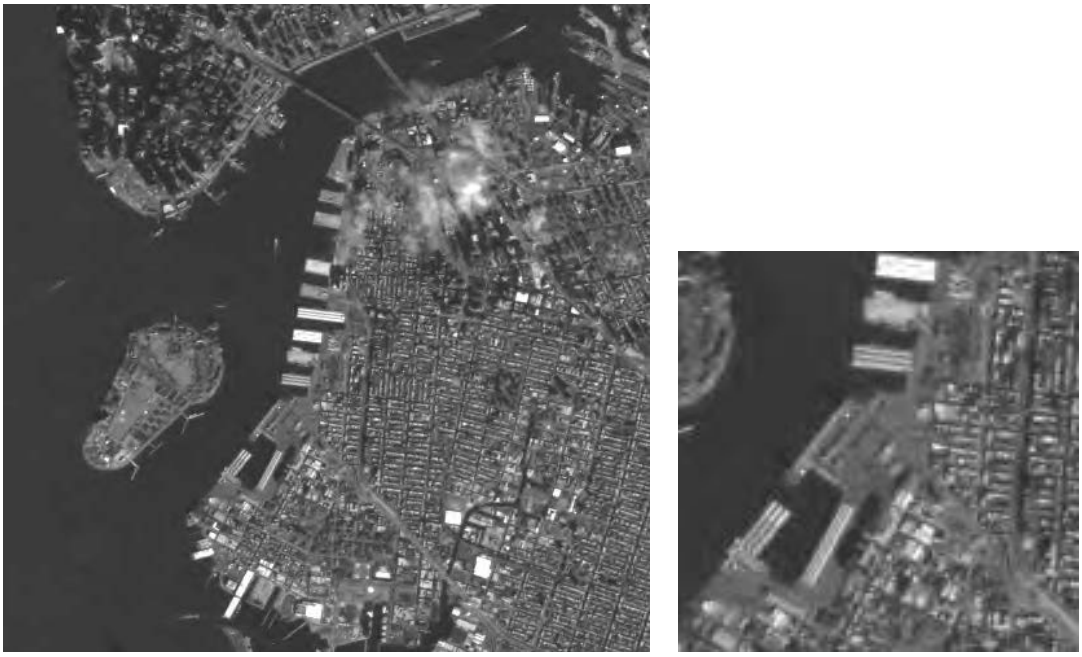
Cependant, le traitement par bloc que nous appliquons pour la représentation et le codage conduit à des effets de blocs, visibles tout particulièrement sur les zones homogènes. On peut également les apercevoir sur des zones à faibles contrastes où de nombreux blocs ne sont représentés que par leur valeur moyenne. Ces effets de blocs sont par exemples visibles sur les vignettes de 150x150 pixels de *Hambourg*. Ces blocs ont été laissés de côté par le codeur au profit de blocs plus contrastés afin d'améliorer la qualité globale de l'image reconstruite. Augmenter le débit serait nécessaire afin de mieux représenter ces blocs. Sur les zones structurées et à forts contrastes, les effets de blocs sont en revanche peu perceptibles grâce à une représentation des blocs de meilleure qualité.

Afin de réduire les effets de blocs, les auteurs dans [XLLZ14] ont opté pour la sélection de patchs avec un certain chevauchement, de manière à améliorer la continuité entre patchs voisins. Mais cela requiert de coder davantage de patchs et dégrade donc les performances débit-distorsion. L'utilisation d'un filtre anti-blocs, comme appliqué dans HEVC, pourrait également être envisagée comme étape de post-traitement sur l'image décodée.



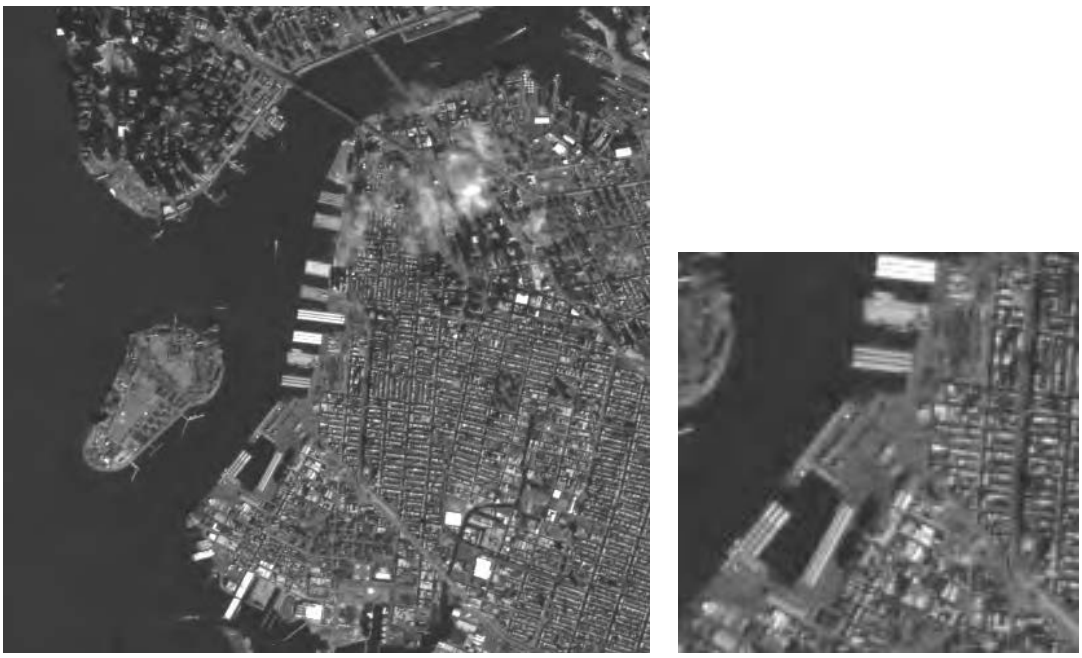
(a) 500x500 pixels (b) 150x150 pixels

FIGURE 4.16 – Image Originale (parties de *New York* : 500x500 (a) et 150x150 (b)).



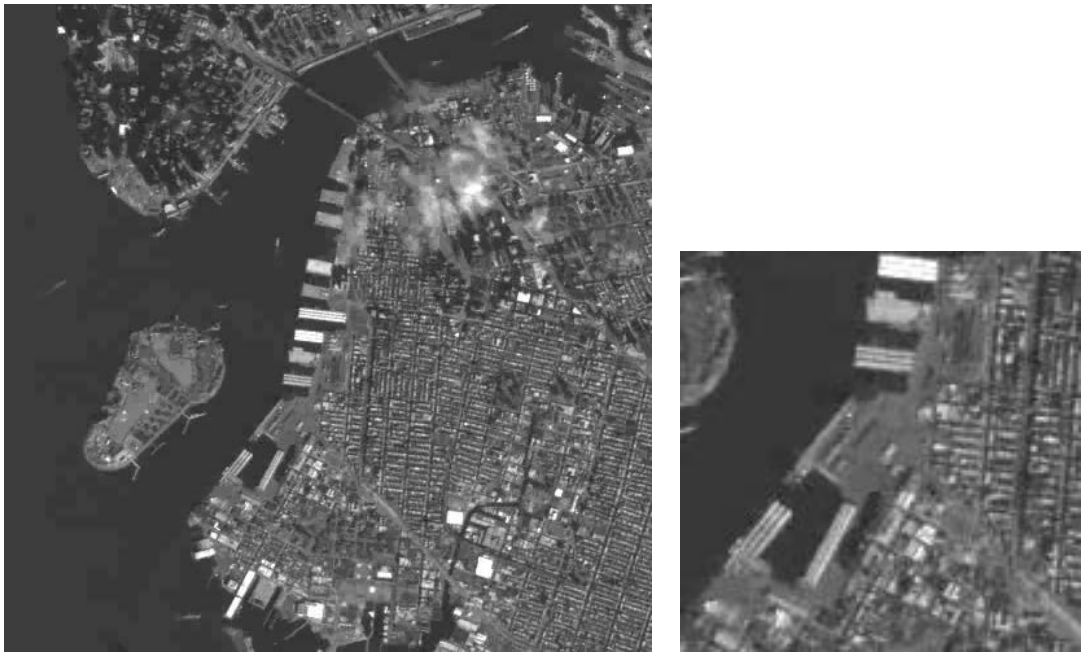
(a) 500x500 pixels (b) 150x150 pixels

FIGURE 4.17 – Image reconstruite par CCSDS 122 à 0.7 bpp (PSNR global sur *New York* de 30.53 dB).



(a) 500x500 pixels (b) 150x150 pixels

FIGURE 4.18 – Image reconstruite par JPEG 2000 (9/7) à 0.69 bpp (PSNR global sur *New York* de 31.18 dB).



(a) 500x500 pixels (b) 150x150 pixels

FIGURE 4.19 – Image reconstruite par le codeur associé à la Structure Adaptative à 0.7 bpp (PSNR global sur *New York* de 30.88 dB).

(a) 500x500 pixels (b) 150x150 pixels

FIGURE 4.20 – Image Originale (parties de *Hambourg* : 500x500 (a) et 150x150 (b)).

(a) 500x500 pixels (b) 150x150 pixels

FIGURE 4.21 – Image reconstruite par CCSDS 122 à 0.7 bpp (PSNR global sur *Hambourg* de 34.58 dB).

(a) 500x500 pixels (b) 150x150 pixels

FIGURE 4.22 – Image reconstruite par JPEG 2000 (9/7) à 0.69 bpp (PSNR global sur *Hambourg* de 35.11 dB).

(a) 500x500 pixels (b) 150x150 pixels

FIGURE 4.23 – Image reconstruite par le codeur associé à la Structure Adaptative à 0.7 bpp (PSNR global sur *Hambourg* de 34.41 dB).

**Utilisation d'une décomposition de type OMP**

La décomposition de type OMP fournissant de meilleurs résultats que celle de type MP en représentation (voir chapitre 3), cette dernière a été intégrée au codeur optimisé. Cela nécessite à chaque nouvel atome sélectionné de recalculer tous les coefficients précédemment calculés pour le bloc courant afin de calculer le rapport $\frac{\Delta D}{\Delta R}$ pour l'optimisation débit-distorsion. Pour cela, la quantification et le codage de tous les coefficients recalculés doivent de nouveau être réalisés, augmentant ainsi la complexité.

Cependant, les résultats obtenus avec cette modification du codeur optimisé ont montré des gains peu significatifs. Il semble que, contrairement à MP, OMP ne puisse pas compenser l'erreur de quantification à cause de sa contrainte d'orthogonalité impliquant la mise à jour de tous les coefficients calculés précédemment. En effet, en intégrant l'erreur de quantification dans la décomposition, on cherche un atome compensant au mieux cette erreur. Mais le re-calcul des coefficients change cette erreur de quantification qui n'est alors plus compensée correctement. Cela rend donc l'intégration de la quantification dans la décomposition quasiment inutile, contrairement à MP. Ainsi, là où le schéma de codage utilisant la décomposition de type OMP présente un gain par rapport à celui utilisant celle de type MP lorsque la quantification n'est pas intégrée dans la décomposition, ce gain devient peu significatif lorsque la quantification est intégrée dans la décomposition. C'est pourquoi la décomposition de type OMP n'est pas appliquée dans ce chapitre pour le cas du codage.

Une autre variante testée a été de combiner les décompositions de type MP et de type OMP au sein du codeur. Le but recherché est de tirer profit de la meilleure décomposition réalisée par OMP tout en cherchant à compenser les erreurs de quantification avec MP. On souhaite donc que des décompositions de type OMP soient réalisées afin de calculer tous les coefficients du bloc, sauf pour le dernier où une décomposition de type MP peut permettre de compenser les erreurs de quantification accumulées, un choix est alors effectué entre MP et OMP.

Mais le nombre de coefficients par bloc n'est pas constant et au moment où un coefficient est calculé, on ne sait pas s'il sera le dernier coefficient calculé pour ce bloc in fine. C'est pourquoi les deux décompositions sont calculées à chaque étape. L'optimisation débit-distorsion sélectionne alors, en plus du bloc auquel ajouter un nouvel atome, la décomposition à réaliser pour calculer le nouveau coefficient, entre MP et OMP. Si à un moment donné, une décomposition de type MP est choisie pour un bloc, il est toujours possible d'ajouter un coefficient à la représentation de ce bloc, mais cette décomposition sera alors modifiée en décomposition de type OMP, avant de choisir le type de la prochaine décomposition, afin qu'une décomposition de type MP ne soit potentiellement présente que pour le dernier coefficient du bloc. Le fait de devoir à chaque étape calculer les deux types de décompositions augmente évidemment la complexité du codeur.

Finalement, ce schéma de codage, plus complexe, apporte un léger gain en débit-distorsion qui augmente lentement avec le débit. Pour *New York* par exemple, ce gain est de 0.01 dB à 0.3 bpp, 0.08 dB à 0.7 bpp et 0.16 dB à 1.2 bpp. Aux mêmes débits, les gains pour *Hambourg* sont respectivement de 0.03 dB, 0.07 dB et 0.15 dB. Ces gains semblent ainsi relativement faibles vis-à-vis de l'augmentation de complexité du schéma de codage, raison pour laquelle cette variante n'est pas appliquée par la suite.

**Taille des blocs**

Le codeur optimisé a également été testé en utilisant des blocs de $16 \times 16$ pixels plutôt que des blocs $8 \times 8$. Cela nécessite d'apprendre des structures de dictionnaires où chaque atome possède 256 valeurs au lieu de 64. Les structures ont dans ce cas été apprises sur 40 niveaux pour $K = 1024$. Les mêmes valeurs de $\Delta_{DC}$ ont été choisies et la même procédure a été appliquée afin de choisir les pas de quantification des coefficients $Q$ optimaux selon les points de débit. Un gain important est observé mais seulement à très bas débit pour 0.1 bpp (+0.9 dB pour *New York*, +0.8 dB pour *Hambourg*, +0.5 dB pour *Los Angeles* et +1.4 dB pour *Désert*), ce qui permet de se rapprocher voire d'atteindre les résultats de JPEG 2000 (9/7) à ce débit, un débit pour lequel les images satellite sont de mauvaise qualité. L'utilisation de blocs de taille $8 \times 8$ donne pour 0.2 bpp des résultats similaires ou supérieurs à ceux obtenus avec des blocs de taille $16 \times 16$, puis supérieurs à partir de 0.3 bpp.

**Taille des dictionnaires**

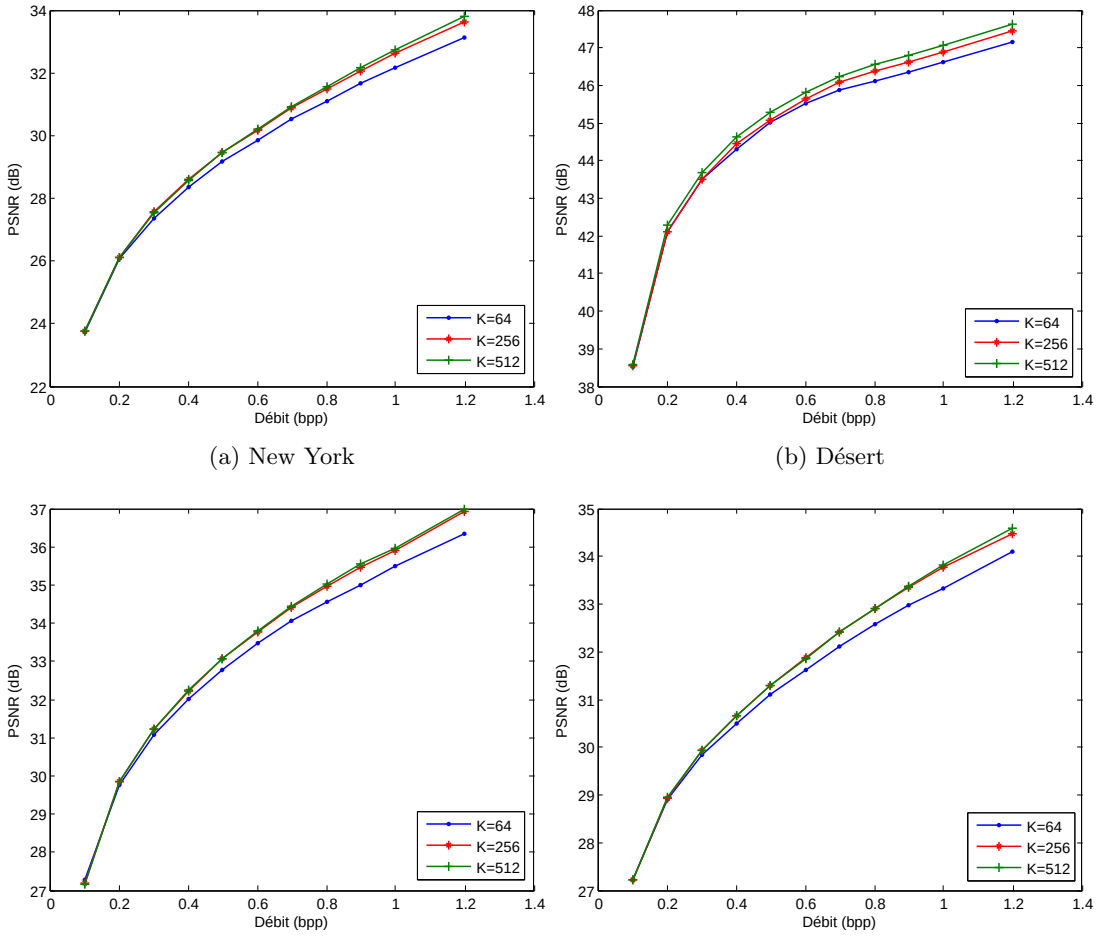
En faisant varier le paramètre $K$ de la taille de chaque dictionnaire au sein de la Structure Adaptative, afin de jouer sur le niveau de redondance des dictionnaires, on s'aperçoit que les résultats s'améliorent avec la redondance du dictionnaire, mais que cette amélioration devient peu significative au dessus de K=256 (Fig. 4.24). Ces résultats concordent avec ceux obtenus dans [BE08] notamment. Les dictionnaires complets (K=64) sont les moins efficaces. En rendant les dictionnaires quatre fois sur-complets (K=256), ils deviennent plus performants et permettent d'obtenir des représentations plus parcimonieuses, ce qui compense l'augmentation du coût de codage des indices, lié à la taille du dictionnaire. Utiliser des dictionnaires huit fois sur-complets (K=512) n'améliore ensuite que légèrement les résultats et seulement à haut débit. En fait, lorsque la taille des dictionnaires devient trop importante, ces derniers risquent de souffrir d'"overfitting", ce qui n'est pas souhaitable. C'est pourquoi on observe sur l'image *Désert*, proche de certaines images d'apprentissage, un gain entre K=256 et K=512 plus important que sur les autres images test. A noter que les dictionnaires de 64 et 256 atomes sont initialisés avec un dictionnaire DCT tandis que ceux de 512 atomes sont initialisés sur des vecteurs d'apprentissage, 512 n'ayant pas de racine carrée entière nécessaire au calcul du dictionnaire DCT 2D (calculé par le produit de Kronecker de deux dictionnaires DCT 1D).

### 4.2.3 Génération d'un flux binaire (*bitstream*) scalable

Une fois la sélection des atomes et le calcul des coefficients, ainsi que leur codage, ont été réalisés d'après le schéma de codage précédemment décrit (Fig. 4.6), le *bitstream* contenant les informations utiles au décodeur peut être créé.

Le *bitstream* conserve les mêmes éléments de base que le *bitstream* d'ITAD [ZGK11] mais ordonnés différemment. Chaque bloc de l'image test est représenté par un code de Huffman pour sa valeur moyenne, codant le résidu de la DPCM, et de plusieurs paires coefficient/indice, le nombre étant variable selon les blocs. Le coefficient est représenté par un code de Huffman et l'indice par un code à longueur fixe. Chaque paire coefficient/indice, ainsi que le code pour la valeur moyenne, sont systématiquement suivis par un *flag* de signalisation 'EOB' de 1 bit indiquant si le codage du bloc est terminé ou non. Ainsi ce code est à $'0'$ lorsque une ou plusieurs paires coefficient/indice sont encore à coder pour le bloc courant et à $'1'$ lorsque toutes les paires ont été codées et donc que le codage du bloc est achevé.

Mais plutôt que d'ordonner le *bitstream* par bloc en représentant l'image bloc après bloc, nous choisissons de l'ordonner par niveau de la Structure Adaptative, c'est à dire en plaçant dans le *bitstream* toutes les paires coefficient/indice d'un niveau donné, avant de passer au niveau suivant. Ainsi, le *bitstream* contient tout d'abord l'ensemble des codes de Huffman correspondant aux valeurs moyennes des blocs, chacun étant suivi du *flag* 'EOB' (1 bit). Puis l'ensemble des paires coefficient/indice de niveau 1 sont placées dans le *bitstream*, chacune suivie de son *flag* 'EOB' (1 bit). Viennent ensuite les paires de niveau 2, puis celles de niveau 3, etc. La parcimonie par bloc étant variable,




(a) New York (b) Désert

(c) Hambourg (d) Los Angeles

FIGURE 4.24 – Courbes débit-distorsion du codeur optimisé associé à la Structure Adaptative, pour différentes tailles des dictionnaires $K$ au sein de la structure.

les niveaux contiendront de moins en moins de paires, jusqu'à ce que toutes les paires soient incluses dans le *bitstream*.

Cette flexibilité du *bitstream* est rendue possible par le codage de chaque coefficient et de chaque indice de façon individuelle. Elle permet d'obtenir une scalabilité en débit. En effet, le *bitstream* peut être arrêté à tout débit $r_b < R_b$, avec $R_b$ le débit cible initial. Les coefficients étant en général plutôt décroissants, les plus grands coefficients, correspondant aux premiers niveaux de la Structure Adaptative et donc aux premières itérations de l'algorithme de décomposition de type MP, sont ainsi codés en priorité. Certes, en coupant le *bitstream* au débit $r_b$, la qualité de reconstruction obtenue pour ce débit n'est pas optimale, car la reconstruction optimale a été calculée pour un débit cible $R_b$ plus grand. En coupant le *bitsream* après $i$ niveaux par exemple, au point de débit $r_b$, il peut arriver de coder un coefficient au niveau $i$ alors que le débit alloué à ce

coefficient aurait été alloué au codage d'un coefficient à un niveau plus profond et d'un autre bloc si le débit cible avait été fixé à $r_b$. De plus le choix du pas de quantification aurait peut-être été différent. Mais cela permet d'obtenir une approximation de l'image décodée avant d'avoir décodé l'intégralité du *bitstream*, en différents points de débit souhaités, et dont la qualité s'améliore au fur et à mesure que le *bitstream* est reçu, pour devenir optimale lorsqu'il a été entièrement décodé.

Cette méthode d'adaptation du débit, consistant simplement à couper le *bitstream* à un débit $r_b$ inférieur au débit cible initial $R_b$, ne présente dans [FiVVF06] (avec un schéma de codage et un dictionnaire différents) qu'une faible perte en PSNR par rapport à un codage optimal avec le débit $r_b$ comme débit cible.

## 4.3 Intégration de dictionnaires structurés appris au sein de HEVC Intra

HEVC [SOHW12] (voir chapitre 2) est un standard de compression de vidéos également très efficace pour la compression d'images fixes en mode Intra. Contrairement à JPEG 2000 transformant les images directement, HEVC utilise de nombreuses prédictions et encode ensuite les résidus de prédiction. Pour cela, une transformée similaire à une DCT entière est appliquée, suivie d'une quantification et d'un codage entropique réalisé par le CABAC.

Nous avons pu voir que les dictionnaires structurés appris permettaient d'obtenir une meilleure qualité de reconstruction qu'un dictionnaire DCT pré-défini.

C'est pourquoi nous allons tenter de modifier l'étape de transformation de HEVC par l'utilisation de décompositions parcimonieuses sur des dictionnaires structurés appris, dans l'espoir d'améliorer cette étape de transformation, tout en conservant les étapes efficaces de prédictions et de codage de HEVC.

Nos tests sont réalisés sur la version 10.0 du HM. Les changements effectués sont exploratoires et ne respectent pas la normalisation de HEVC.

### 4.3.1 Remplacement de l'étape de transformation

La transformation au sein de HEVC, similaire à une transformée DCT entière, est remplacée dans nos expérimentations par des décompositions parcimonieuses sur des dictionnaires structurés appris. Les étapes de transformation et transformation inverse sont donc modifiées, pour les blocs de luminance et seulement en mode Intra étant donné que nous nous contentons de coder une image fixe. Les autres étapes de prédiction, quantification ou codage entropique ne sont pas modifiées.

La transformation au sein de HEVC, appliquée sur les TU carrées de tailles 4x4, 8x8, 16x16 ou 32x32, est séparée en deux transformations 1D utilisant des matrices de transformations de taille 4x4, 8x8, 16x16 ou 32x32 selon la taille de la TU. Ces matrices de transformation sont des approximations de matrices DCT arrondies sur des entiers de 8 bits (signe inclus). Elles sont entières et ne sont pas normalisées, mais respectivement de normes $2^7$, $2^{7,5}$, $2^8$ et $2^{8,5}$ environ sur leurs lignes et colonnes. Pour

les blocs 4x4 de luma en mode Intra, une approximation d'une DST est appliquée à la place de la DCT.

Concernant le "scaling" réalisé au niveau de l'étape de transformation inverse au décodeur, une division par $2^7$ avec arrondi est réalisée entre les deux transformations inverses 1D, puis une division par $2^{12}$ (dans le cas d'images 8 bits) avec arrondi est réalisée suite aux deux transformations inverses 1D.

Tentons tout d'abord de reproduire les résultats de HEVC mais en utilisant des dictionnaires DCT 2D et DST 2D normalisés complets et OMP pour le calcul des coefficients à la place de la transformée DCT (et DST) de HEVC et de ses "scalings". Les blocs 8x8, 16x16 et 32x32 utilisent ainsi respectivement les dictionnaires DCT 2D complets de taille 64x64, 256x256 et 1024x1024, tandis que les blocs 4x4 utilisent un dictionnaire DST 2D complet de taille 16x16. A noter que réaliser une transformation DCT 2D sur un bloc $n \times n$ ou appliquer OMP pour une parcimonie de $n^2$ atomes sur un dictionnaire DCT 2D complet de taille $n^2 \times n^2$ permet de calculer les mêmes coefficients de transformée.

Dans le but de retrouver les mêmes coefficients décodés que HEVC, le fait d'utiliser des dictionnaires normalisés implique de devoir multiplier les coefficients décodés par le carré de la norme des lignes des matrices de transformation de HEVC. De plus, la division par $2^7$ et $2^{12}$ doit également être réalisée. Ainsi, on retrouve les coefficients décodés obtenus par le HEVC original en divisant les coefficients obtenus dans notre cas par 32, 16, 8 et 4, pour les blocs 4x4, 8x8, 16x16 et 32x32 respectivement. A l'encodeur, les coefficients issus de la décomposition par OMP sur les dictionnaires normalisés doivent donc être multipliés par 32, 16, 8 et 4, respectivement pour les blocs 4x4, 8x8, 16x16 et 32x32.

Afin d'accélérer les expérimentations, la taille des TU a été limitée aux tailles de blocs 4x4 et 8x8, les transformations sur les dictionnaires pouvant autrement être relativement coûteuses en temps de calcul. De plus, une portion de *New York* de 512x512 pixels est utilisée comme image test.

Ainsi, en utilisant les dictionnaires DCT et DST normalisés, avec pour OMP une parcimonie de 16 pour les blocs 4x4 et de 64 pour les blocs 8x8, on parvient à retrouver les résultats du HEVC original (Fig. 4.25). A noter que les différents points des courbes débit-distorsion sont obtenus en modifiant la valeur du paramètre Qp de quantification de HEVC.

### 4.3.2 Expérimentations

Nous proposons désormais de remplacer les dictionnaires DCT et DST par des dictionnaires appris. Les dictionnaires sont appris sur les résidus de prédiction de HEVC des images d'apprentissage (*Boston18*, *Champagne18*, *Correze20*, *Dubai18*, *Libye4*, *MontSaintMichel18*, *NewYork2*, *NewYork4* et *Sochaux19*, voir Annexe 1), obtenus pour un Qp de 32. 100 000 vecteurs sont utilisés pour apprendre chaque dictionnaire. Pour les blocs 4x4, un dictionnaire K-SVD complet de taille 16x16 est appris pour une parcimonie de 10 atomes, en l'initialisant avec un dictionnaire DST. Une structure en branche, pour sa simplicité et son efficacité, est également apprise, chaque dictionnaire

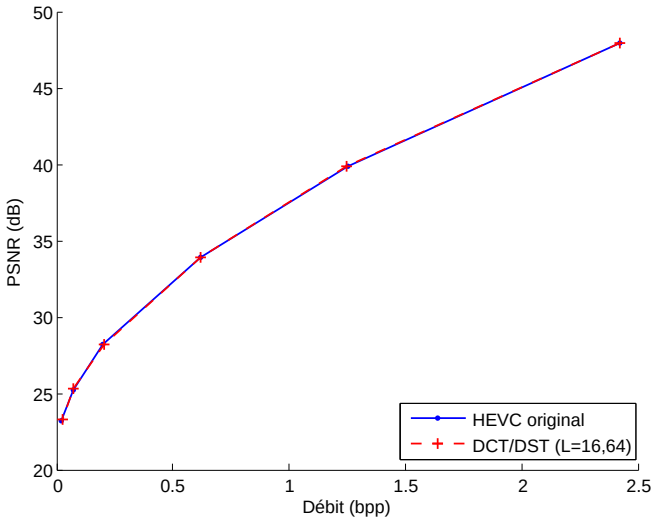



FIGURE 4.25 – Reproduction des résultats de HEVC avec des dictionnaires DCT/DST normalisés et OMP, pour une parcimonie (L) de 16 pour les blocs 4x4 et 64 pour les blocs 8x8.

au sein de la branche étant de taille 16x16 et initialisé avec un dictionnaire DST. Pour les blocs 8x8, le dictionnaire K-SVD complet de taille 64x64 est appris pour une parcimonie de 20 atomes et est initialisé avec un dictionnaire DCT. Au sein de la structure en branche apprise, chaque dictionnaire est de taille 64x64 et est initialisé par un dictionnaire DCT. Les dictionnaires K-SVD sont appris en 50 itérations. Pour les structures en branche, 50 itérations sont utilisées pour appendre le premier niveau, puis 10 pour les niveaux suivants.

**Performances de dictionnaires appris par rapport à des transformées prédéfinies**

Cherchons tout d'abord à vérifier l'efficacité et donc l'intérêt de l'apprentissage sur les résidus de prédictions de HEVC, par rapport à l'utilisation d'un dictionnaire prédéfini pour la transformation.

Suite aux décompositions réalisées sur les différents dictionnaires, pour une parcimonie de 10 atomes pour les blocs 4x4 et de 20 pour les blocs 8x8, les coefficients calculés sont rangés en début de bloc. Plutôt que de les ranger selon leur indice, les coefficients sont donc regroupés, le reste du bloc étant constitué de valeurs nulles. On compare ainsi l'efficacité des décompositions en codant les coefficients uniquement, sans coder leur position, c'est-à-dire sans prendre en compte le codage des indices (transmis à part sans codage). Pour chaque bloc, les coefficients obtenus sur la structure en branche sont rangés selon leur niveau, c'est-à-dire dans l'ordre dans lequel ils sont calculés, tandis que ceux obtenus sur les dictionnaires "plats" (DST, DCT et K-SVD) sont rangés par ordre décroissant.

L'arrangement des coefficients dans le bloc est réalisé afin de faciliter l'encodage

du CABAC. L'arrangement s'adapte en fait à la méthode de parcours utilisée par le CABAC. Ce parcours est réalisé par sous-bloc de taille 4x4 de façon diagonale, horizontale ou verticale selon le mode de prédiction Intra choisi [SB12]. C'est pourquoi, en correspondance à ces méthodes de parcours du CABAC, les coefficients sont arrangés par sous-bloc 4x4 de manière horizontale pour les modes de prédiction 22 à 30, verticale pour les modes de prédiction 6 à 14 et diagonale pour les autres modes [LBH+12]. A noter que cela n'est appliqué que pour les blocs 4x4 et 8x8, les blocs 16x16 et 32x32 (non utilisés dans nos expérimentations) n'utilisent qu'une méthode de parcours diagonale.

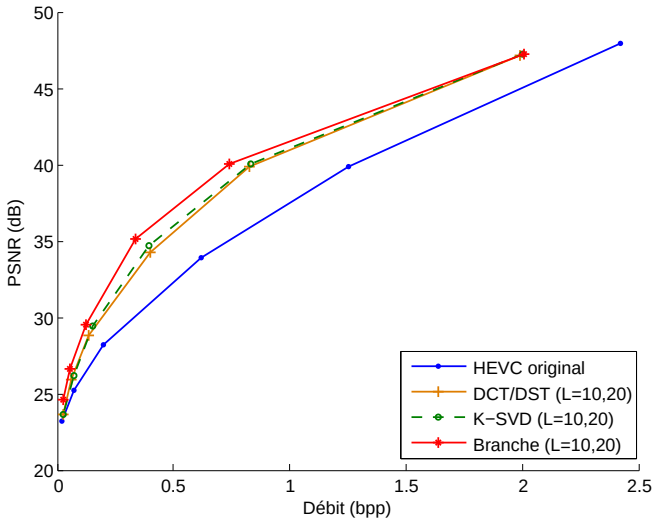



FIGURE 4.26 – Comparaisons débit-distorsion selon les dictionnaires intégrés à HEVC, sans le codage des indices, pour une parcimonie (L) de 10 pour les blocs 4x4 et 20 pour les blocs 8x8.

Dans ces conditions, la structure en branche permet d'obtenir une meilleure représentation que les dictionnaires K-SVD et DCT/DST (Fig. 4.26). Le dictionnaire K-SVD est certes moins efficace que la branche mais est tout de même meilleur que les dictionnaires DCT/DST. A haut débit (faible Qp), les courbes se rejoignent. L'apprentissage, réalisé sur les résidus de prédiction pour un Qp de 32, semble perdre en efficacité lorsque le Qp devient trop faible. Les courbes se retrouvent nettement au-dessus de celle du HEVC original car les indices ne sont pas codés dans cette expérimentation. On déduit de cette expérimentation que l'utilisation de dictionnaires appris sur les résidus de prédiction de HEVC, et tout particulièrement de dictionnaires structurés, présente un intérêt vis-à-vis de dictionnaires prédéfinis.

### Prise en compte du codage des indices

Nous cherchons ensuite à coder, en plus des coefficients, les indices correspondants. Pour cela, les coefficients sont positionnés dans le bloc codé par le CABAC selon leur indice correspondant. Les indices sont ainsi codés en codant la position des coefficients dans le bloc, comme cela est fait dans HEVC.

Cependant, cela impose dans notre cas certaines contraintes dues à la structuration en branche. Tout d'abord, un bloc ne peut pas utiliser dans sa décomposition deux atomes au même indice, même s'ils sont sélectionnés à des niveaux différents de la branche. En effet, les deux coefficients se retrouveraient dans ce cas à la même position dans le bloc. Plus la décomposition descend dans la branche et plus cela devient contraignant, réduisant le choix des atomes à chaque niveau. Cette contrainte pénalise donc particulièrement les résultats de la branche à haut débit. Ensuite, cet ordonnancement oblige à sauvegarder le niveau dans la branche correspondant à chaque coefficient, ces derniers n'étant plus rangés dans l'ordre des niveaux auxquels ils sont sélectionnés. Cela entraîne donc un surplus d'informations à coder par rapport aux dictionnaires "plats".

De plus, contrairement à une transformée DCT ordonnançant les coefficients selon la fréquence de la fonction d'onde DCT correspondante, les coefficients calculés sur la branche n'ont pas de raison d'être davantage regroupés en début de bloc et peuvent être dispersés à travers le bloc, rendant leur codage par le CABAC plus lourd. Cela est vrai également pour K-SVD.

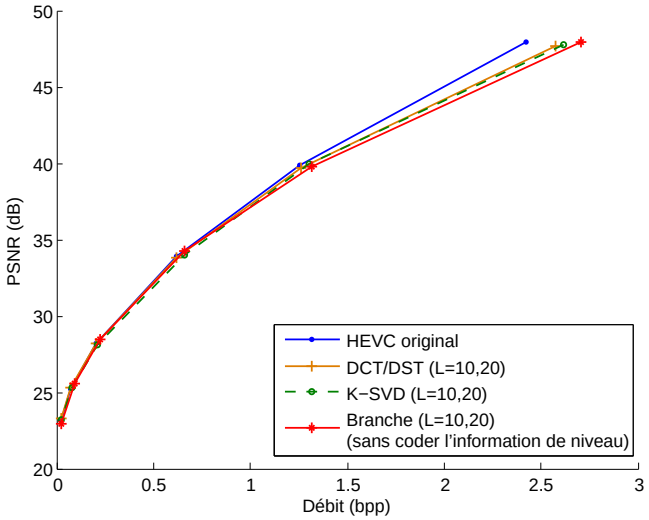



FIGURE 4.27 – Comparaisons débit-distorsion selon les dictionnaires intégrés à HEVC, pour une parcimonie (L) de 10 pour les blocs 4x4 et 20 pour les blocs 8x8.

En ajoutant le codage des indices, grâce à l'ordonnancement des coefficients dans le bloc selon leur indice, on s'aperçoit que la structure en branche est fortement pénalisée par les contraintes précédemment énoncées (Fig. 4.27), alors même que le codage des niveaux des coefficients n'est pas pris en compte dans cette expérimentation. Les résultats de la structure en branche sont alors similaires à ceux des dictionnaires DCT/DST, et passent même en-dessous à haut débit, la structure en branche étant alors particulièrement pénalisée par la contrainte interdisant pour un bloc de sélectionner deux atomes au même indice. Réaliser une décomposition de type OMP sur la structure en branche, à la place de la décomposition de type MP, améliorerait probablement les résultats, mais cela ne modifierait pas les contraintes imposées à la structure en branche.

On note également que la limitation de la parcimonie affecte les résultats à haut débit, la quantification y étant plus fine. A bas débit, c'est la quantification plus grossière qui limite le nombre de coefficients non nuls codés par bloc davantage que la limite de parcimonie.

Il apparaît donc que, bien que la structure en branche permette d'obtenir une représentation de meilleure qualité que les dictionnaires prédéfinis, cette structure est inadaptée à la structuration des données et au codage réalisé au sein de HEVC, et nécessiterait ainsi de modifier HEVC en profondeur.

## 4.4 Spécialisation des dictionnaires dans le cadre de l'observation persistante

Suite à l'aparté sur HEVC, nous allons désormais tenter de rendre l'apprentissage des dictionnaires structurés plus efficace en les spécialisant à une scène particulière dans le cadre de l'observation persistante.

### 4.4.1 Cadre de l'observation persistante

L'objectif est ici de se placer dans un cas d'étude quasiment idéal où l'image test est très proche des images d'apprentissage afin de rendre l'apprentissage et donc les dictionnaires très efficaces.

Pour cela, deux séquences d'observation persistante sont utilisées : *Ville* (Fig. 4.28) et *Rade* (Fig. 4.29). Ces séquences d'images représentent une scène dont le fond est quasiment statique et où certains objets sont en mouvement d'une image à l'autre, tels que des voitures, des arbres ou encore de l'eau.

On souhaite alors étudier les performances en représentation que peuvent atteindre la Structure Adaptative et la Structure en branche, puis en codage avec le codeur optimisé et la Structure Adaptative, dans ce cas précis très favorable où les dictionnaires sont spécialisés à une scène précise.

### 4.4.2 Expérimentations

Les deux séquences d'images, *Ville* (Fig. 4.28) et *Rade* (Fig. 4.29), sont des séquences de 194 et 160 images respectivement. Les images sont initialement des images couleurs dont chaque composante est représentée sur 8 bits. Nous travaillons ici sur la composante Y de luminance, calculée à partir des composantes de Rouge, de Vert et de Bleu. La différence entre la dernière et la première image de chaque séquence (Fig. 4.30) montre que les différences majeurs sont les voitures en mouvement sur *Ville* et les bateaux sur *Rade*. On retrouve tout de même des structures sur les images de différence indiquant que le fond n'est pas exactement statique entre les images.

Les structures sont apprises sur 10 niveaux, chaque dictionnaire en leur sein possédant 256 atomes (K=256). Comme précédemment, 50 itérations sont appliquées au premier niveau, puis 10 aux niveaux suivants.

FIGURE 4.28 – Séquence *Ville* (dernière image).

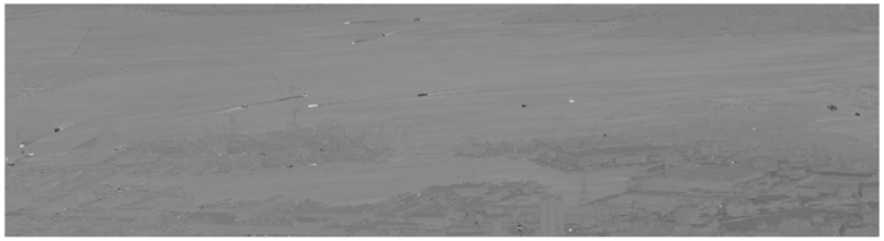
FIGURE 4.29 – Séquence *Rade* (dernière image).

Les premières images de chaque séquence sont utilisées comme images d'apprentissage et la dernière image comme image test. L'image test comme les images d'apprentissage sont découpées en blocs 8x8 ne se chevauchant pas de façon à créer les patchs de test et d'apprentissage respectivement. Afin de rendre l'apprentissage le plus efficace possible, les patchs sont sélectionnés sur les images d'apprentissage aux mêmes emplacements que sur l'image test, c'est-à-dire que la même grille d'échantillonnage des patchs est utilisée à l'apprentissage et lors du test.

#### 4.4.2.1 Tests de représentation en fonction de la parcimonie

Pour ces tests de représentation mesurant la qualité de reconstruction en fonction de la parcimonie, deux structures sont comparées : la Structure Adaptative et la structure en branche. Trois ensembles d'apprentissage sont créés. Le premier est composé des dix premières images de la séquence *Ville*, le deuxième des dix premières images de la séquence *Rade* et le troisième des cinq premières images des deux séquences. Les deux structures sont apprises sur ces trois ensembles de façon à apprendre des dictionnaires

(a) Séquence *Ville*



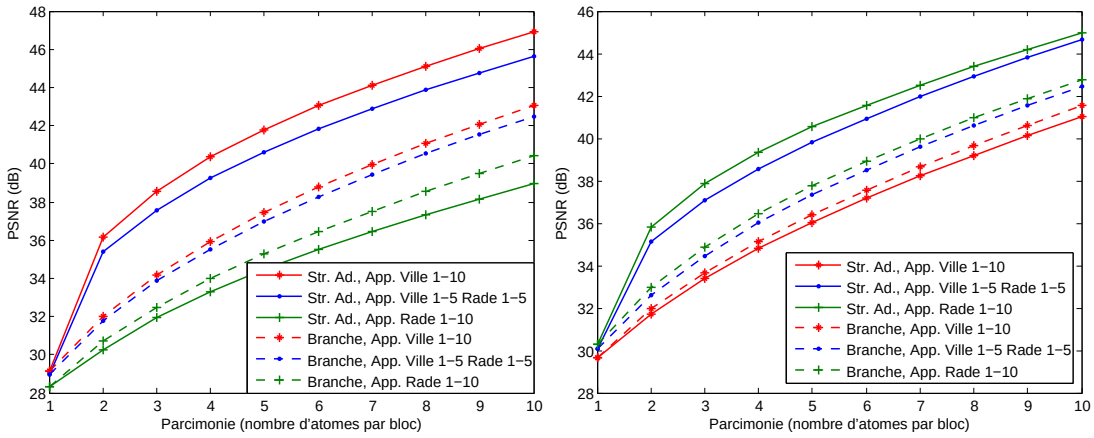
(b) Séquence *Rade*

FIGURE 4.30 – Images de différences (ramenées entre 0 et 255) entre la dernière et la première image de chaque séquence.

spécialisés pour la séquence *Ville*, pour la séquence *Rade*, ou bien mixtes car appris sur des images provenant des deux séquences.

Les deux images de test, la dernière image de *Ville* et la dernière image de *Rade*, sont ensuite décomposées sur ces six structures de dictionnaires pour différentes valeurs de parcimonie (Fig. 4.31).

Dans ce cas particulier où l'image test est très proche des images d'apprentissage, la Structure Adaptative se distingue particulièrement de la structure an Branche. En effet, grâce à sa structure composée de nombreux dictionnaires, la Structure Adaptative est capable de davantage se spécialiser que la structure en branche, limitée à un unique dictionnaire par niveau. Les deux courbes en tête correspondent ainsi aux Structures Adaptatives apprises d'abord sur les images de la même séquence que l'image test, puis sur la base d'apprentissage mixte comprenant des images des deux séquences. On retrouve ensuite les structures en branche apprises sur ces deux bases d'apprentissage




(a) Séquence *Ville* (dernière image) (b) Séquence *Rade* (dernière image)

FIGURE 4.31 – Tests PSNR-parcimonie pour les dernières images des deux séquences, sur différents dictionnaires appris.

dans le même ordre. Concernant le cas défavorable où l'image test d'une séquence est décomposée sur une structure apprise sur les images de l'autre séquence, la structure en branche est alors plus performante, de part sa plus grande généricité.

On remarque que les courbes sont plus resserrées pour *Rade* que pour *Ville*, ce qui est sans doute dû au fait que l'image contient une zone relativement homogène de grande taille, plus facile à représenter que des zones texturées.

**Sensibilité à la grille d'échantillonnage des patchs**

Nous souhaitons maintenant étudier l'impact de la sélection des patchs d'apprentissage sur la séquence *Ville*. Plutôt que de les choisir sur la même grille de patchs que l'image test sur les dix premières images de la séquence (176 000 patchs sont sélectionnés dans ce cas), 500 000 patchs sont sélectionnés à des emplacements aléatoires sur ces mêmes images.

On s'aperçoit alors que les résultats chutent fortement, en particulier pour la Structure Adaptative, lorsque les patchs d'apprentissage et de test ne correspondent plus (Fig. 4.32), l'apprentissage étant alors moins spécialisé aux patchs de l'image test. L'apprentissage est donc sensible à la sélection des patchs au sein des images et nécessite un alignement des images d'apprentissage et de test pour obtenir les meilleurs résultats.

#### 4.4.2.2 Tests de codage

Nous proposons désormais d'appliquer le codeur optimisé pour les dictionnaires structurés dans le cas le plus favorable, c'est-à-dire lorsque l'image test est décomposée sur un dictionnaire structuré appris sur les images de la même séquence avec la Structure Adaptative. De plus, les patchs de l'image test et des images d'apprentissage sont

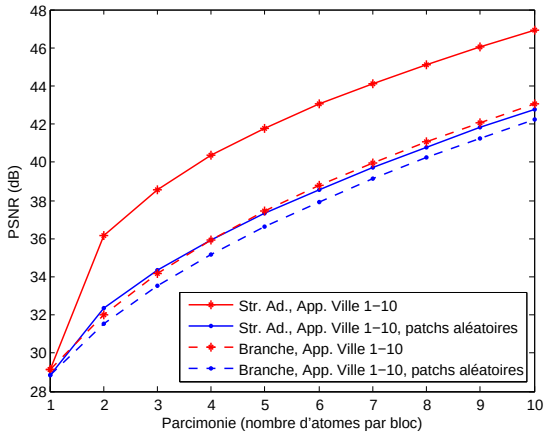
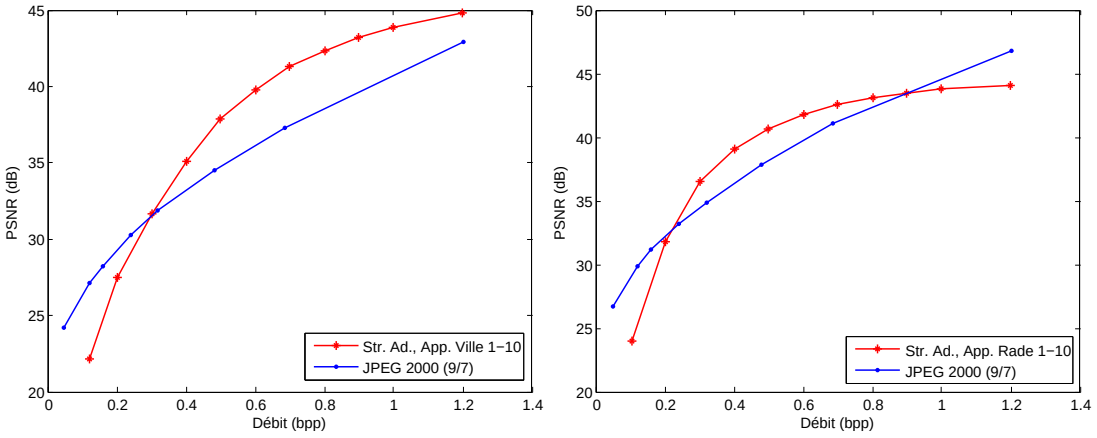



FIGURE 4.32 – Comparaisons en représentation avec un apprentissage sur des patchs aléatoirement sélectionnés.

sélectionnés sur la même grille de patchs, c'est-à-dire aux mêmes emplacements.




(a) Séquence *Ville* (dernière image) (b) Séquence *Rade* (dernière image)

FIGURE 4.33 – Tests de codage pour les dernières images des deux séquences, avec le codeur optimisé autour de la Structure Adaptative.

Dans ce cas d'étude presque idéal, le codeur surpasse alors JPEG 2000 (avec l'ondelette 9/7) à partir de 0.2-0.3 bpp, le gain pouvant même atteindre plusieurs dB (Fig. 4.33). A haut débit, les résultats ont tendance à saturer rapidement par rapport à JPEG 2000, qui peut alors repasser en tête, car les structures ont été apprises sur 10 niveaux seulement, limitant à 10 le nombre d'atomes sélectionnés par bloc. Augmenter le nombre de niveaux permettrait d'augmenter encore les performances, en particulier à haut débit. Le fait que la courbe sature plus vite pour *Rade* doit être dû au fait que l'image contient beaucoup de zones d'eau peu texturées sur lesquelles l'augmentation du nombre d'atomes dans la décomposition a peu d'impact. De plus, le $\Delta_{DC}$ a été fixé

à 1 mais pourrait être augmenté, ce qui améliorerait probablement encore davantage les résultats, en particulier à bas débit.

Dans le cas où 500 000 patchs d'apprentissage sont sélectionnés à des emplacements aléatoire sur les dix premières images de *Ville*, plutôt qu'aux mêmes emplacements que les patchs de test, on observe comme en représentation une chute importante des résultats (Fig. 4.34).

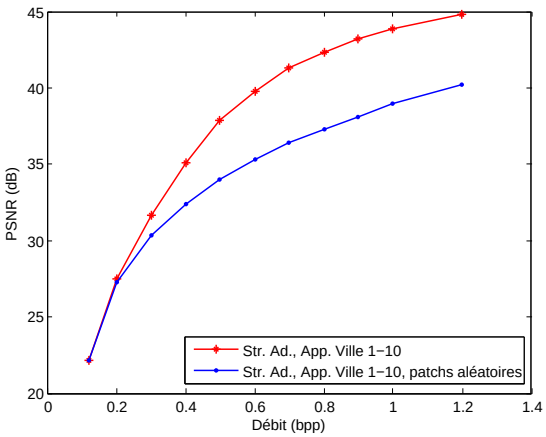



FIGURE 4.34 – Comparaisons en codage avec un apprentissage sur des patchs aléatoirement sélectionnés.

En conclusion, en se plaçant dans un cas quasiment idéal où l'image test est très proche des images d'apprentissage, et en alignant ces images, on obtient un apprentissage performant permettant de surpasser JPEG 2000 de façon significative.

Dans ce cas où l'on souhaite spécialiser le dictionnaire à une scène en particulier, la Structure Adaptative est plus adaptée que la structure en branche, car capable de davantage se spécialiser grâce au nombre de dictionnaires qu'elle peut contenir.

## 4.5 Conclusion

Suite aux comparaisons, réalisées dans le chapitre précédent, des différentes structures de dictionnaires en qualité de reconstruction en fonction de la parcimonie, les comparaisons ont été effectuées dans ce chapitre du point de vue du codage. Les dictionnaires ont d'abord été intégrés dans un schéma de codage simple encodant les valeurs moyennes des blocs ainsi que les coefficients et indices résultant de la décomposition. Ont ainsi été constatées les bonnes performances des dictionnaires structurés que sont la Structure Adaptative, la structure en "cerf-volant" et la structure en branche, par rapport aux dictionnaires dits "plats".

Puis le codeur a été optimisé selon une version améliorée du codeur ITAD. Une optimisation débit-distorsion permet de sélectionner les blocs auxquels ajouter de nou-

veaux atomes dans la représentation. De plus, des tables de Huffman sont apprises afin de coder les coefficients. Par rapport à ITAD, l'apprentissage des tables de Huffman et le codage des valeurs moyennes des blocs ont été améliorés, et la quantification intégrée dans les décompositions. Le codeur optimisé, associé à la Structure Adaptative, permet ainsi de surpasser les résultats du codeur ITAD.

Le codeur a ensuite été comparé à des codeurs standards. Il permet d'obtenir de meilleurs résultats débit-distorsion que JPEG et est capable de surpasser CCSDS 122 et JPEG 2000 5/3 sur une certaine plage de débit (typiquement entre 0.2 et 0.6 bpp). Selon la proximité de l'image test avec les données d'apprentissage, il peut s'approcher voire atteindre les résultats de JPEG 2000 9/7. Le codeur peut ainsi pour certains débits rivaliser avec des codeurs standards optimisés, avec un codage entropique simple comparé à celui de JPEG 2000 par exemple.

Dans le but d'améliorer l'étape de transformation réalisée au sein de HEVC, les dictionnaires structurés appris ont par la suite été intégrés à HEVC en remplacement de la transformation de type DCT. Nous avons alors montré que, bien que les dictionnaires structurés appris sur les résidus de prédiction de HEVC permettent d'offrir une représentation de meilleure qualité que les dictionnaires prédéfinis DCT/DST, ils sont soumis à des contraintes, liées à la méthode de codage appliquée dans HEVC, pénalisant leurs résultats.

Enfin, les dictionnaires ont été spécialisés à des scènes particulières dans le cadre de l'observation persistante afin de rendre l'apprentissage plus performant. La Structure Adaptative, capable de davantage se spécialiser, a alors montré de meilleures performances que la structure en branche. De plus, des gains significatifs (jusqu'à 4 dB) ont été observés par rapport à JPEG 2000, dans ce cadre favorable où l'image test est très proche des images d'apprentissage.

Nous avons ainsi pu voir dans ce chapitre que les performances de codage des dictionnaires dépendent fortement de la qualité de l'apprentissage et de la proximité entre les images tests à compresser et les images formant la base d'apprentissage. Même en traitant spécifiquement des images satellites, ces dernières pouvant présenter une grande disparité entre elles, nous n'avons pu que nous rapprocher de JPEG 2000. Il a fallu fortement spécialiser les dictionnaires à des scènes précises pour dépasser JPEG 2000 dans le cadre de l'observation persistante. Bien sûr, le schéma de codage étudié présente un codage entropique simple en comparaison de celui de JPEG 2000, optimisé, et nécessiterait d'être encore amélioré. En particulier, le coût de codage des indices est très pénalisant et le diminuer est un enjeu du codage par représentations parcimonieuses.

Dans la troisième partie de cette thèse, d'autres applications des dictionnaires structurés appris, orientées classifications, vont être étudiées. Une première concernera la reconnaissance de FTM et la seconde traitera de classification supervisée pour de la reconnaissance de scènes.

# Troisième partie

# Apprentissage de dictionnaires structurés pour la classification d'images satellites

# Chapitre 5

# Application des dictionnaires structurés à l'estimation de FTM

## 5.1 Contexte et état de l'art

La FTM (Fonction de Transfert de Modulation) est un moyen d'évaluer les performances en matière de résolution spatiale d'un système d'imagerie [Wil98], c'est-à-dire son niveau de préservation des détails spatiaux.

La FTM peut être définie comme le module de la transformée de Fourier de la PSF ("Point Spread Function"), ou de la LSF ("Line Spread Function") en se restreignant à une direction :

$$FTM = |DFT(PSF)| \tag{5.1}$$

La PSF correspond à la réponse d'un système optique à une source ponctuelle d'illumination. La forme de la PSF donne une indication de la qualité du système : moins elle s'étend et plus le système est de bonne qualité. La LSF correspond quant à elle à la réponse à une ligne de fort contraste. Elle peut être obtenue en dérivant l'ESF ("Edge Spread Function"), qui est la réponse du système à un contour de fort contraste.

La FTM d'un système d'imagerie optique représente donc un indice de performance du système vis-à-vis de la qualité des images produites. Par extension, la FTM mesure ainsi la qualité d'une image : plus la FTM est faible et plus l'image est floue, les détails spatiaux étant moins bien préservés.

La FTM d'un système d'imagerie est fréquemment spécifiée par sa valeur de FTM à la fréquence de Nyquist. La fréquence de Nyquist est définie comme la plus haute fréquence sinusoïdale pouvant être représentée par un signal échantillonné et est égale à la moitié de la fréquence d'échantillonnage du système.

Les mesures de la FTM instrument sont réalisées au sol avant le lancement, mais l'évaluation et le suivi des performances du système sont également nécessaires une fois le satellite en orbite. En effet, les images acquises par des satellites peuvent être dégradées suite à un dérèglement des instruments à bord, survenu par exemple lors du décollage. La mesure de la FTM permet alors de mesurer cette dégradation des images et de détecter un possible dérèglement des instruments à bord. Mise à part

cette estimation de contrôle, il peut être intéressant d'être capable d'estimer la FTM instrument d'un satellite tiers sur lequel aucun contrôle n'est possible.

On cherche donc dans ce chapitre à estimer la FTM d'un système optique, c'est-à-dire la FTM associée à une image test de FTM inconnue.

Une méthode actuellement utilisée est la photographie d'une mire connue située à un endroit précis sur Terre. Mais cela oblige à positionner le satellite sur une zone précise pour réaliser la mesure de la FTM.

L'idée d'utiliser un projecteur dirigé vers le satellite a été explorée dans [LDR94]. L'image prise par le satellite du projecteur correspond presque à la PSF du système et, associée à un modèle de FTM lié au satellite SPOT3 utilisé, elle permet d'effectuer une mesure de la FTM.

Une autre possibilité explorée dans [BLF13] consiste à mesurer la FTM par l'acquisition d'étoiles, intéressantes du fait qu'elles représentent une source ponctuelle immobile pendant la prise de vue. Mais cela nécessite de choisir une zone du ciel riche en étoiles suffisamment brillantes, par exemple la constellation des Pléiades lors des mois d'hiver, et de réaliser plusieurs acquisitions successives de la même zone du ciel afin d'avoir assez d'images d'étoiles. Une FTM moyenne de l'instrument peut alors être calculée grâce à ces images d'étoiles.

Des images d'étoiles sont utilisées dans [FEYM09] également, en utilisant ces sources ponctuelles de lumières pour estimer la PSF du système. La FTM est ensuite évaluée d'après l'estimation de la PSF.

Dans [WLL09], les auteurs proposent une méthode de mesure automatique de la FTM à partir d'images de la Terre. Cette méthode est basée sur la détection de lignes droites en utilisant la transformée de Hough. Puis, après sélection de certaines de ces lignes, la FTM est estimée par une méthode basée sur des forts contours (contours à forts contrastes). En effet, les forts contours permettent de calculer une ESF, puis une LSF qui correspond à la différentielle d'une ESF. Enfin, une transformation de Fourier de la LSF permet de retrouver la FTM. Mais si la méthode échoue à détecter ces lignes droites au sein d'une image, alors l'estimation de la FTM est impossible pour cette image.

Les forts contours sont également utilisés dans [XZ12], où l'analyse d'objets spéciaux est effectuée afin de retrouver la FTM. La méthode est aussi adaptée aux contours non alignés avec la matrice de pixels produite par le dispositif d'imagerie.

Dans [NB01], des images à forts contours, ainsi que des images de pont, prises par l'instrument Hyperion lancé en 2000, sont utilisées pour le calcul de l'ESF ou de la LSF, desquelles la FTM est déduite.

La recherche de contours à fort contraste est également nécessaire dans [Koh04], afin de déterminer l'ESF, puis d'en déduire la LSF et la FTM. L'accès à des mires pouvant être limité, des contours appropriés sont identifiés sur des images opérationnelles classiques. Ainsi, ces contours sont observés aussi bien sur des mires que sur des zones urbaines. Mais si les zones urbaines peuvent offrir davantage de contours et donc davantage de mesures, la difficulté est alors de trouver des contours convenables respectant des contraintes d'orientation (par rapport aux axes du système), de longueur, de contraste

ou encore de niveau de bruit.

Ainsi, la PSF, la LSF et l'ESF peuvent être utilisées afin de calculer la FTM. L'utilisation d'une mire représente la solution la plus utilisée, mais se limite aux satellites présentant une faible GSD (*Ground Sample Distance* : distance au sol entre les centres de deux pixels adjacents), la mire devenant trop petite pour les satellites avec une GSD plus grande. Les patterns naturels peuvent alors être utilisés.

On citera enfin [DLRV03], où les auteurs estiment la FTM d'images inconnues à l'aide d'un réseau de neurones appris sur des images dont le contenu en terme de paysages et de FTM est connu.

Notre objectif est de pouvoir estimer la FTM instrument à partir d'une image quelconque, sans avoir à photographier une mire, à s'orienter vers certaines étoiles ou à rechercher et viser certains objets spécifiques au sol, tels que des ponts ou de forts contours. Pour cela, nous allons apprendre puis utiliser des dictionnaires structurés. Contrairement aux méthodes apprenant un réseau de neurones unique, nous allons apprendre un dictionnaire par valeur de FTM, et sur des scènes variées. C'est alors la qualité de reconstruction relative entre les différents dictionnaires qui va nous permettre d'estimer la FTM. On cherche de cette façon à s'affranchir du contenu des images d'apprentissage par rapport à celui des images test.

Dans ce chapitre, les valeurs de FTM sont exprimées à la fréquence de Nyquist. Elles sont normalement normalisées entre 0 et 1 mais exprimées ici avec un facteur multiplicatif de 100 pour faciliter la lecture. Une FTM de valeur 35 correspond donc en réalité à une FTM de valeur 0.35.

De plus, les images d'apprentissage comme de test sont des images simulées pour différentes valeurs de FTM.

## 5.2 Expérimentations préliminaires

L'objectif de ces expérimentations préliminaires est de tester le pouvoir d'apprentissage d'un certain niveau de flou, selon la valeur de FTM, des dictionnaires structurés et d'étudier leur capacité de discrimination entre différentes FTM dans le but de les utiliser dans le cadre de la reconnaissance de FTM.

La Structure Adaptative [AMGL13b] (voir Chapitre 3) est utilisée pour sa capacité d'apprentissage car nous pensons qu'elle a le pouvoir d'apprendre des niveaux de flou en plus des informations de texture, de par le nombre de dictionnaires qu'elle est capable de contenir, tout en gardant une complexité de l'algorithme de décomposition comparable à celle d'un petit dictionnaire "plat". Cinq Structures Adaptatives sont donc apprises, jusqu'au niveau 20, sur une même base d'apprentissage composée de 8 images variées (*Boston18, Champagne18, Correze20, Dubai18, Libye4, NewYork2, NewYork4* et *Sochaux19*, présentées en Annexe 1), mais pour des valeurs de FTM différentes (5, 10, 15, 20 et 35). Chaque dictionnaire est ainsi spécifique à une valeur de FTM. A noter que les dictionnaires sont appris sur des blocs 8x8 des images sans retirer la moyenne, afin que toute l'information soit prise en compte, et que les Structures Adaptatives

contiennent des dictionnaires complets de 64 atomes.

Deux scènes test (*New-York* (Fig. 3.3) et *Mataro* (Fig. 5.1)), disponibles pour les mêmes valeurs de FTM (ce qui représente donc 10 images test), sont ensuite décomposées sur ces cinq dictionnaires structurés (1 atome est sélectionné par niveau avec la décomposition de type MP), en sélectionnant de 1 à 20 atomes par bloc pour la reconstruction. Les PSNR des images reconstruites sont alors calculés, pour chaque valeur de parcimonie, afin de construire pour chaque image test une courbe PSNR-parcimonie par dictionnaire. A noter que *New-York* possède toujours une zone commune présentant le même découpage des blocs, sur ses 200 dernières lignes, avec une image d'apprentissage.

(a) FTM=35 (b) FTM=5

FIGURE 5.1 – Image test : *Mataro* (1024x1024) pour deux valeurs de FTM : FTM=35 (a) et FTM=5 (b). Images 8 bits en niveaux de gris.

En observant les PSNR des images reconstruites pour des parcimonies allant de 1 à 20 atomes sélectionnés par bloc, pour l'image *New York* avec une FTM de 5 par exemple (Fig. 5.2), on se rend compte que les résultats de décomposition sur les différents dictionnaires sont relativement proches et donc peu discernables, en particulier lorsque le nombre d'atomes est faible. On décide donc de zoomer sur la plage de 15 à 20 atomes sélectionnés, c'est à dire sur les résultats de décomposition allant au plus profond de la Structure Adaptative, où les courbes se distinguent davantage les unes des autres. Chaque graphique au sein des figures, 5.3 pour *New York* et 5.4 pour *Mataro*, correspond à une image test d'une certaine FTM, décomposée sur les cinq dictionnaires puis reconstruite.

Deux cas de figure se distinguent alors. Dans le premier cas, une courbe se démarque des autres, la FTM correspondante est alors la FTM de l'image test (Fig. 5.3 (a)(b)), Fig. 5.4(a)). Dans le second cas, plusieurs courbes se distinguent des autres mais sont

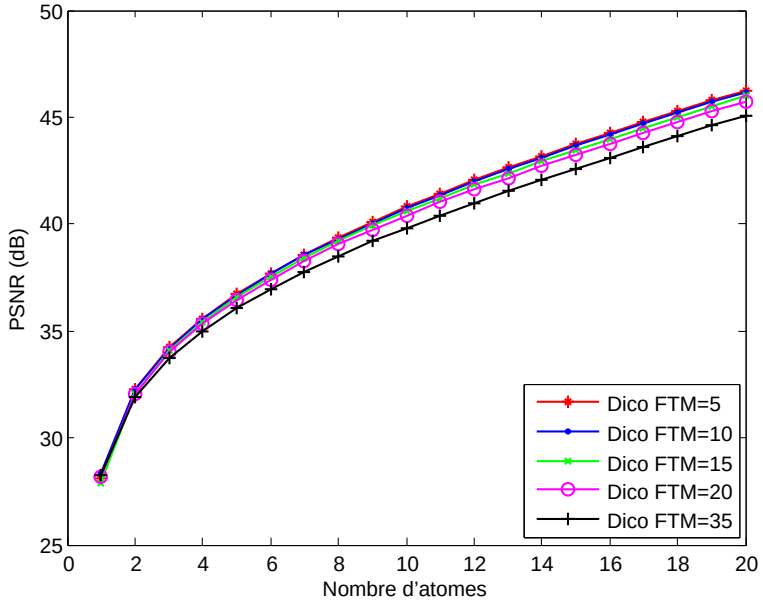
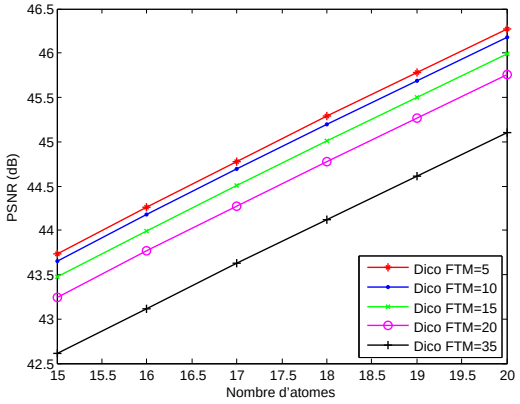


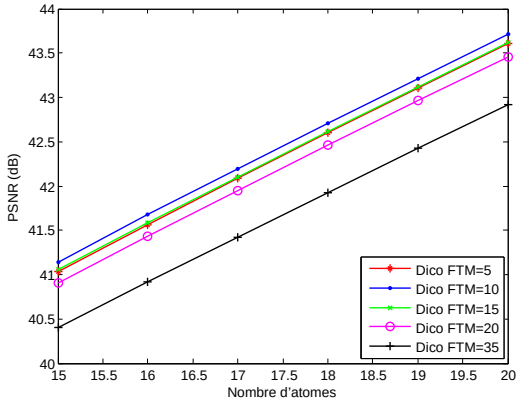

FIGURE 5.2 – Tests sur *New York*, FTM=5.

entre-elles très proches (Fig. 5.3 (c)(d)(e)), Fig. 5.4(b)(c)(d)). Dans ce cas-là, plusieurs dictionnaires représentent l'image test quasiment aussi bien, car il semble qu'ils aient appris suffisamment de flou. C'est alors parmi ces courbes celle correspondant à la plus grande FTM qui donne la FTM de l'image test. Par exemple, dans le cas de la figure 5.3(c), les dictionnaires de FTM 10 et 15 donnent les meilleurs résultats et des résultats très similaires. La FTM de l'image test est donc de 15, car si elle était de 10, le dictionnaire de FTM 15 n'aurait pas appris suffisamment de flou pour avoir une aussi bonne représentation, et on se retrouverait alors dans le cas de la figure 5.3(b). Pour ces 9 images de test, on peut ainsi fixer un seuil afin de déterminer quelles courbes sont proches de la courbe donnant le PSNR maximal, et parmi les FTM retenues, choisir la FTM la plus grande comme estimation de la FTM de l'image test. En se plaçant à une parcimonie de 20 atomes, un seuil de 0.08 dB permettrait d'estimer correctement la FTM de ces 9 images test. En revanche, on commettrait une erreur pour la figure 5.4(e) où la courbe du dictionnaire de FTM 35 est trop éloignée de la meilleur courbe obtenue avec le dictionnaire de FTM 20. Ce cas incite à utiliser des seuils variables plutôt qu'un seuil fixe, de façon à s'adapter à l'écart entre les FTM comparées. En effet, on peut être plus tolérant sur le seuil lorsque l'écart entre les FTM comparées est plus important, comme ici entre les FTM 20 et 35. Nous reviendrons sur la détermination de ces seuils dans la section suivante.

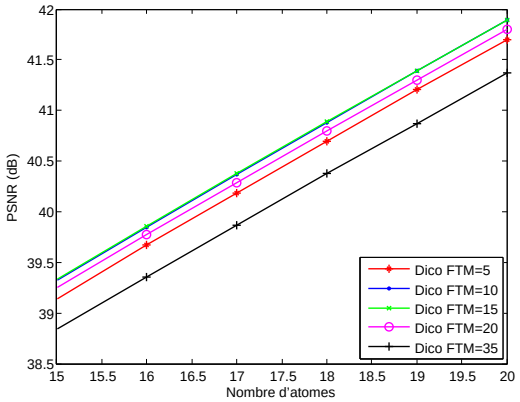
Il semble au vu de ces résultats qu'une image d'une FTM donnée soit mieux représentée par un dictionnaire ayant appris certaines caractéristiques de la texture de l'image ainsi que le niveau de flou suffisant. Ainsi, une image test avec une certaine FTM doit être de préférence représentée par un dictionnaire de FTM égale ou inférieure, de telle sorte qu'il ait appris suffisamment de flou. Cependant un autre critère



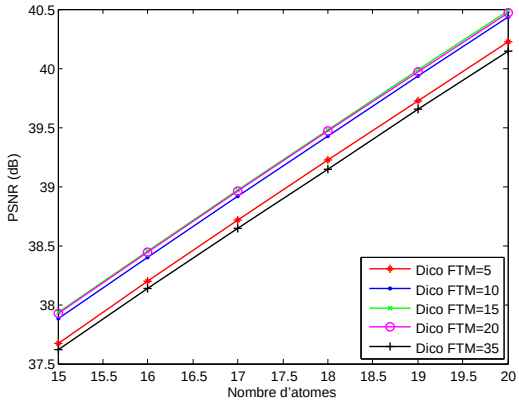

(a) FTM=5



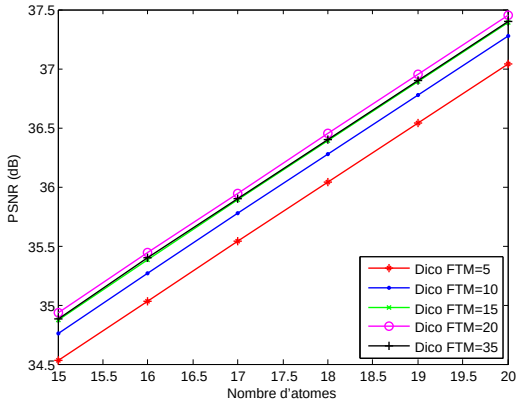

(b) FTM=10




(c) FTM=15

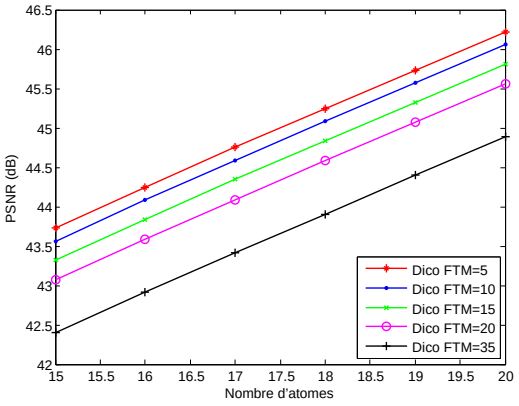



(d) FTM=20

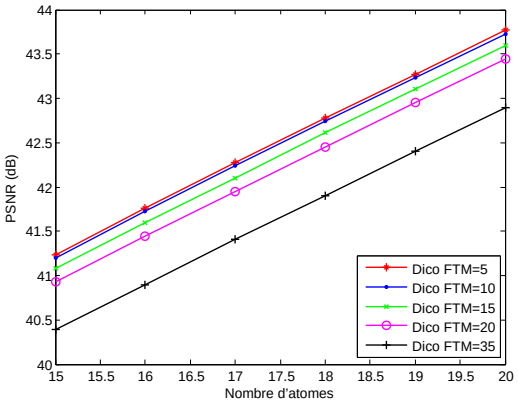



(e) FTM=35

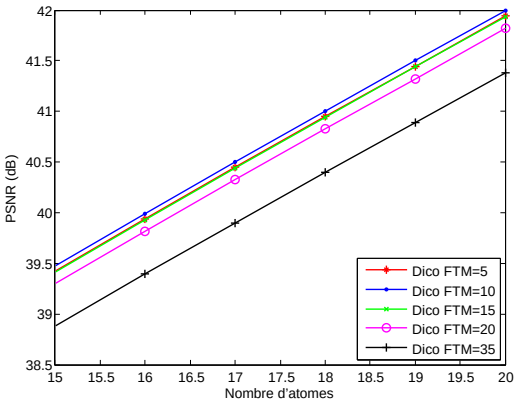
FIGURE 5.3 – Tests sur *New York*.



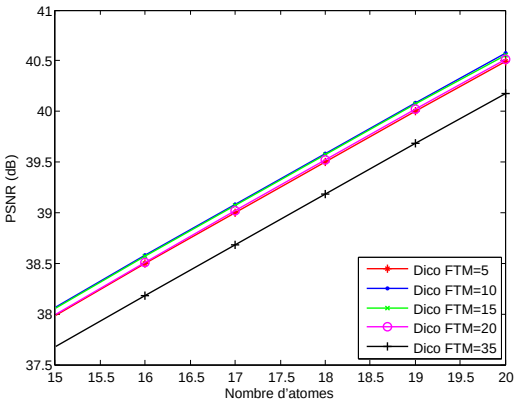

(a) FTM=5



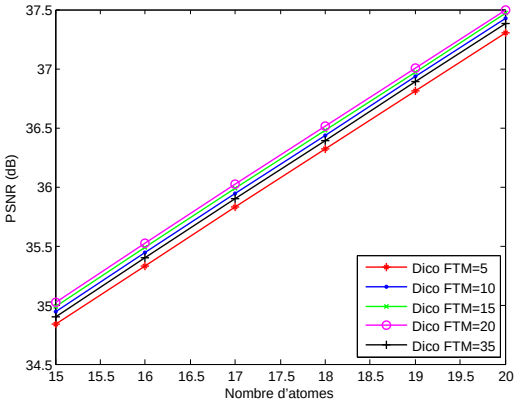

(b) FTM=10




(c) FTM=15




(d) FTM=20




(e) FTM=35

FIGURE 5.4 – Tests sur *Mataro*.

se dégage : l'écart entre la FTM test et celle du dictionnaire. En effet, un dictionnaire de FTM trop éloignée de celle de l'image test, même si sa FTM est inférieure et que l'on peut considérer qu'il a appris suffisamment de flou, donnera des résultats inférieurs vis-à-vis du dictionnaire de FTM optimale. En conclusion, deux critères semblent influencer les résultats : le niveau de flou appris par le dictionnaire, qui doit être suffisant, et l'écart entre la FTM de l'image test et celle du dictionnaire, qui doit rester limité.

D'après les résultats obtenus, nous élaborons donc l'hypothèse suivante : un dictionnaire à structure adaptative de FTM inférieure mais proche de celle de l'image test, ayant appris suffisamment de flou, peut donner des résultats très similaires à ceux obtenus avec le dictionnaire de FTM égale à celle de l'image test, contrairement à un dictionnaire de FTM supérieure, ou de FTM inférieure mais trop éloignée. Cette hypothèse nous sera utile dans la section suivante afin de construire la méthode d'estimation de FTM.

## 5.3 Description de la méthode

La méthode d'estimation de FTM que nous présentons ici est basée sur des techniques d'apprentissage de dictionnaires sur des images d'entraînement de FTM connues et variées. De multiples dictionnaires sont ainsi appris, correspondant chacun à un niveau de FTM différent. Un algorithme de décision permet ensuite d'utiliser ces dictionnaires appris afin de fournir une estimation de la FTM d'une image test. C'est l'adéquation relative de ces dictionnaires à décrire l'image qui est utilisée pour estimer la FTM, afin de s'affranchir autant que possible du contenu des images d'apprentissage. L'algorithme de décision se base sur l'utilisation de seuils de décision, qui peuvent également être appris.

### 5.3.1 Apprentissage de dictionnaires

L'apprentissage des dictionnaires nécessite une base d'images d'entrainement variées, c'est-à-dire représentant plusieurs types de scènes (ville, champs, mer, forêt...) et disponibles pour différentes valeurs de FTM connues.

La Structure Adaptative (voir Chapitre 3), permettant d'apprendre de nombreux dictionnaires et pouvant atteindre une profondeur importante, est utilisée. Une Structure Adaptative est ainsi apprise sur la base variée des images d'entrainement, pour chaque valeur de FTM disponible, de telle sorte à apprendre une structure par FTM. De cette façon, chaque dictionnaire structuré est adapté à une valeur de FTM particulière, mais pas à une scène particulière.

### 5.3.2 Algorithme de décision

#### Principe

Afin d'estimer la FTM d'une image test, on commence par la décomposer bloc par bloc pour une certaine parcimonie fixe sur chacun des dictionnaires appris, puis on calcule le PSNR des images reconstruites. On obtient ainsi une valeur de PSNR par FTM ($PSNR_i$). La valeur de PSNR maximale ($PSNR_{maxi}$) est alors recherchée. Si cette valeur se démarque suffisamment des autres valeurs de PSNR, alors la FTM correspondante est l'estimation de la FTM de l'image test. Si en revanche, plusieurs valeurs de PSNR sont proches de la valeur maximale du PSNR, alors c'est parmi les FTM correspondantes ($ftm_{cand}$) la FTM la plus grande qui est choisie comme estimation de la FTM de l'image test. Ce choix est basé sur l'hypothèse émise suite aux tests préliminaires selon laquelle un dictionnaire à structure adaptative de FTM inférieure mais proche de celle de l'image test, ayant donc appris suffisamment de flou, peut donner des résultats très similaires à ceux obtenus avec le dictionnaire de FTM égale à celle de l'image test, contrairement à un dictionnaire de FTM supérieure.

#### Algorithme

L'algorithme de décision appliqué à une image test de FTM inconnue suit les étapes suivantes :

Soit $ftm$ l'ensemble des $N$ valeurs de FTM disponibles : $ftm = \{ftm_i, i = 1..N\}$.

(i) Décomposition de l'image test sur les différents dictionnaires (pour une valeur de parcimonie fixée) et calcul des PSNR des images reconstruites :

$$PSNR = \{PSNR_i, i = 1..N\} \tag{5.2}$$

(ii) Détermination de la valeur maximale de PSNR parmi les valeurs calculées :

$$PSNR_{maxi} = \max(PSNR) \tag{5.3}$$

(iii) Calcul des écarts entre chaque valeur de PSNR et la valeur maximale :

$$\Delta PSNR_i = PSNR_{maxi} - PSNR_i, \; pour \; i = 1..N \tag{5.4}$$

(iv) Détermination de l'ensemble des FTM candidates contenant les FTM dont l'écart de PSNR correspondant est inférieur à un seuil fixé :

$$ftm_{cand} = \{ftm_i | \Delta PSNR_i <= seuil, i = 1..N\} \tag{5.5}$$

(v) Choix de la FTM maximale parmi l'ensemble des FTM candidates comme estimation de la FTM de l'image test :

$$FTM_{estimee} = \max(ftm_{cand}) \tag{5.6}$$

Un seuil est donc nécessaire à l'étape (iv) afin de déterminer si l'écart entre le PSNR maximal ($PSNR_{maxi}$) et chaque PSNR ($PSNR_i$) est suffisamment faible pour que la FTM correspondante soit gardée comme candidate potentielle, ou si cet écart est trop important, la FTM correspondante étant alors rejetée.

### 5.3.3 Détermination des seuils de décision

Comme pour les dictionnaires, on cherche à apprendre le seuil utilisé dans l'algorithme de décision, grâce à un deuxième ensemble d'images d'apprentissage composé de "scènes" dont chacune est disponible pour différentes valeurs de FTM (chaque couple scène-valeur de FTM donne ainsi une image). Pour un seuil donné, il est possible d'estimer son efficacité sur ces nouvelles images d'apprentissage en appliquant l'algorithme de décision expliqué précédemment, afin d'obtenir une FTM estimée pour chaque image, à comparer à sa véritable FTM. On cherche alors de façon empirique le seuil ($seuil_{opt}$) minimisant la somme sur les différentes images (toutes les images d'apprentissage des seuils, quelle que soit leur FTM) de la valeur absolue de l'erreur entre la FTM estimée et la vraie FTM :

$$seuil_{opt} = \underset{seuil}{\arg\min} \sum_{images} |FTM_{vraie} - FTM_{estimee_{seuils}}| \tag{5.7}$$

La première idée serait d'utiliser un seuil fixe. Mais un même écart entre deux valeurs de PSNR n'a pas la même signification s'il est observé pour deux FTM proches ou deux FTM éloignées. C'est pourquoi nous avons choisi d'opter pour des seuils croissants avec l'écart entre les FTM comparées, les FTM comparées correspondant à la FTM de PSNR maximal ($PSNR_{maxi}$) et la FTM de chaque PSNR ($PSNR_i$). Cela permet d'augmenter le seuil et donc la tolérance lorsque des FTM éloignées sont comparées et d'avoir une comparaison plus fine avec un seuil plus faible pour deux FTM proches.

Ces seuils sont modélisés sous la forme d'une fonction affine croissante, choisie pour sa simplicité :

$$y = ax + b \tag{5.8}$$

avec $y$ le seuil associé à l'écart $x$ entre les FTM comparées.

La recherche empirique des seuils optimaux ($seuils_{opt}$) résulte ainsi en un couple $(a_{opt}, b_{opt})$ correspondant respectivement au coefficient directeur et à l'ordonnée à l'origine de la fonction affine des seuils optimaux.

On ajoute enfin une certaine tolérance afin de conserver plusieurs couples $(a, b)$ correspondant aux meilleures fonctions de seuils, au lieu de garder uniquement le couple $(a_{opt}, b_{opt})$ et donc une seule fonction de seuils. On garde ainsi les seuils vérifiant :

$$SAD_{\substack{seuils \\ (a,b)}} \leq SAD_{\substack{seuils_{opt} \\ (a_{opt},b_{opt})}} + \lfloor \frac{x}{100} SAD_{\substack{seuils_{opt} \\ (a_{opt},b_{opt})}} \rfloor \tag{5.9}$$

avec $x$ le pourcentage de tolérance et

$$SAD_{\substack{seuils \\ (a,b)}} = \sum_{images} |FTM_{vraie} - FTM_{estimee_{\substack{seuils \\ (a,b)}}}| \tag{5.10}$$

A noter que dans cette expression, la somme est réalisée sur toutes les images utilisées pour la détermination des seuils, quelle que soit leur FTM. De plus, dans cette formule, la SAD est calculées sur des valeurs de FTM pour les différentes images, et non sur des pixels comme cela est usuellement le cas.

Pour chaque fonction de seuils conservée, nous avons également calculé pour chaque scène la pire erreur (en valeur absolue) obtenue entre la FTM estimée et la véritable FTM, pour les différentes valeurs de FTM disponibles. La somme de ces pires erreurs sur les différentes scènes est alors calculée pour chaque fonction de seuils conservée, et la plus faible somme indique la fonction de seuils finalement choisie (fonction affine de coefficient directeur $a_f$ et d'ordonnée à l'origine $b_f$), donnant les $seuils_{finaux}$ :

$$\underset{(a_f,b_f)}{seuils_{finaux}} = \underset{\substack{seuils \\ (a,b)}}{\arg\min} \sum_{scenes} \max_{ftm} |FTM_{vraie} - \underset{(a,b)}{FTM_{estimee_{seuils}}}| \tag{5.11}$$

## 5.4 Expérimentations

Pour ces expérimentations, nous disposons de 13 images (8 bits de taille $2400 \times 2400$) différentes (voir Annexe 2), chacune disponible pour 22 valeurs de FTM (sur une échelle de 0 à 100) : 5, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38 et 40. L'estimation de la FTM est donc plus complexe que lors des expérimentations préliminaires puisque les FTM peuvent être très proches les unes des autres rendant les images quasiment identiques à l'œil.

Huit images sont utilisées comme base d'apprentissage (*Hambourg2*, *NewYork1*, *NewYork2*, *NewYork4*, *SalonProvence1*, *SalonProvence2*, *SalonProvence4* et *Toulouse2*) et permettent d'apprendre, comme pour les expérimentations préliminaires, un dictionnaire à structure adaptative par valeur de FTM. Là encore, des patchs 8x8 sont extraits de ces images d'apprentissage, sans en retirer la moyenne, afin de former les vecteurs d'entraînement des dictionnaires. Les Structures Adaptatives sont ainsi apprises sur 400000 vecteurs d'entraînement chacune, jusqu'à 20 niveaux de profondeur, et composées de dictionnaires complets de 64 atomes.

Les cinq images restantes sont utilisées comme images de tests et ont pour nom *Hambourg1, LosAngeles1, NewYork3, SalonProvence3* et *Toulouse1*. Ces images sont décomposées sur les dictionnaires avec une parcimonie de 20 atomes et tirent donc profit de toute la profondeur des Structures Adaptatives. Les photos étant parfois prises sur des zones voisines (mais pas forcément au même instant), des images de tests peuvent présenter sur leurs bords des parties communes, c'est-à-dire représentant les mêmes zones, avec des images d'apprentissage. C'est ainsi le cas de *Hambourg1, NewYork3, SalonProvence3* et *Toulouse1*. Mais cela est le cas pour toutes les valeurs de FTM dans les comparaisons.

Pour la détermination des meilleurs seuils, la recherche empirique est effectuée pour des valeurs de $a$ et $b$ variant entre 0 et 1 avec un pas de 0.001 pour $a$ et de 0.01 pour $b$. La tolérance permettant de retenir plusieurs seuils est fixée à 5% de la valeur minimale de la somme sur toutes les images (quelle que soit leur FTM) de la valeur absolue de la différence entre la FTM estimée et la véritable FTM. Le choix des seuils finaux est alors réalisé parmi les seuils conservés grâce aux mesures des pires erreurs, comme expliqué à la section précédente.

En appliquant l'algorithme de détermination des seuils de décision sur une ou plusieurs images de test, on obtient un couple $(a_f, b_f)$ donnant les paramètres de la fonction

affine de seuils $y = a_f x + b_f$ associant un seuil à chaque écart entre deux FTM comparées, compris pour ces tests entre 1 et 35. En appliquant l'algorithme plusieurs fois tout en changeant le nombre et la combinaison des images tests utilisées, plusieurs couples ressortent majoritairement :

(1) $(a_f ; b_f) = (0.002 ; 0.05)$
(2) $(a_f ; b_f) = (0.002 ; 0.04)$
(3) $(a_f ; b_f) = (0.003 ; 0.04)$
(4) $(a_f ; b_f) = (0.003 ; 0.03)$
(5) $(a_f ; b_f) = (0.001 ; 0.05)$
(6) $(a_f ; b_f) = (0.001 ; 0.06)$

En observant les fonctions de seuils correspondants à ces différents couples (Fig. 5.5), on remarque que les courbes avec les plus grands coefficients directeurs ont également les plus faibles ordonnées à l'origine, et vice versa.

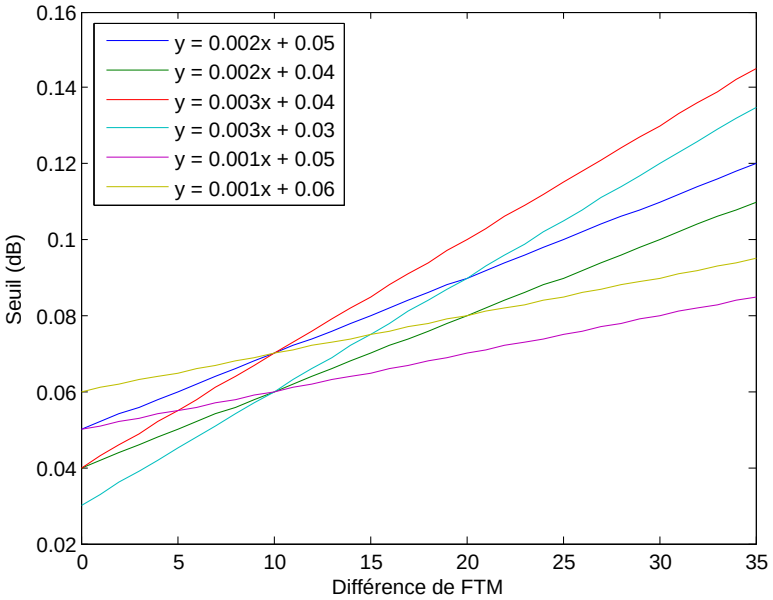



FIGURE 5.5 – Seuils (en dB) déterminés par les images test.

Maintenant que des seuils efficaces ont été déterminés, l'algorithme de décision est appliqué aux 5 images tests, pour toutes leurs valeurs de FTM. On peut ensuite, pour chaque seuil, comparer pour chaque image les 22 FTM estimées par rapport aux vraies FTM.

On remarque, en appliquant les seuils (1) par exemple (Fig. 5.6 - 5.10), l'intérêt de l'algorithme de décision utilisé pour l'estimation de la FTM, par rapport à une simple sélection de la FTM correspondant au meilleur PSNR. Sur ces courbes, chaque point correspond à un test indépendant réalisé sur une image test pour une valeur de FTM particulière. Notre méthode permet d'obtenir des estimations restant proches des véritables valeurs de FTM (vérité terrain), tandis qu'une simple estimation de la FTM

par la recherche du PSNR maximal n'est valable que pour de faibles valeurs de FTM.

Pour les 6 seuils proposés, nous avons reporté les résultats d'estimation de FTM des 5 images, en moyennant les écarts d'estimations obtenus pour les 22 valeurs de FTM disponibles pour chaque image (Tab. 5.1 - 5.6). Le Tableau 5.7 fournit un résumé des résultats obtenus en appliquant l'algorithme de décision avec les 6 seuils différents pour estimer la FTM des images test.

Notre méthode d'estimation de FTM permet donc d'estimer de manière assez juste puisque l'écart entre la FTM estimée et sa véritable valeur est en moyenne de 2.01 sur l'ensemble des images test et des 6 seuils testés.

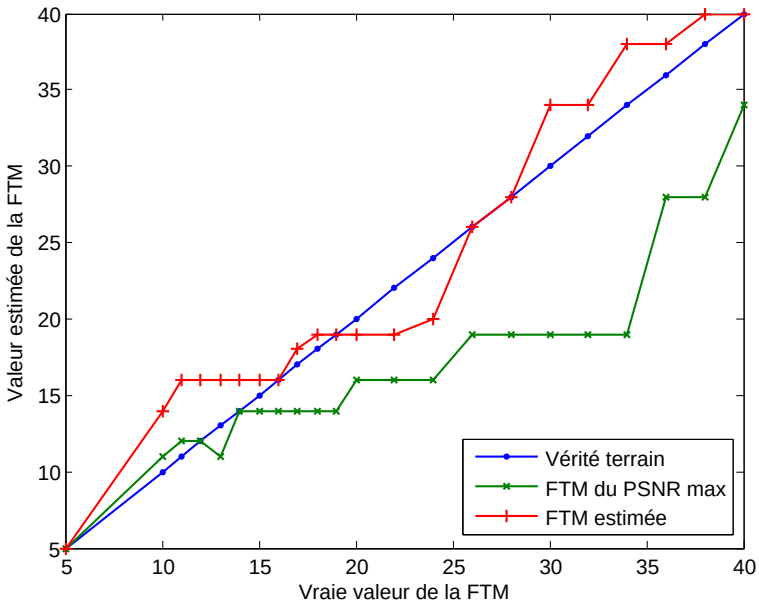



FIGURE 5.6 – Valeurs de FTM estimées pour *Hambourg1* avec les seuils (1).

| Image test | Écart moyen de FTM | Écart min | Écart max |
|---|---|---|---|
| Hambourg1 | 1.95 | 0 | 5 |
| LosAngeles1 | 2.32 | 0 | 6 |
| NewYork3 | 1.45 | 0 | 4 |
| Toulouse1 | 2.50 | 0 | 5 |
| SalonProvence3 | 1.55 | 0 | 4 |
| Moyenne | 1.95 | 0 | 4.80 |

TABLE 5.1 – Estimation de la FTM avec les seuils (1) $(a_f; b_f) = (0.002; 0.05)$.

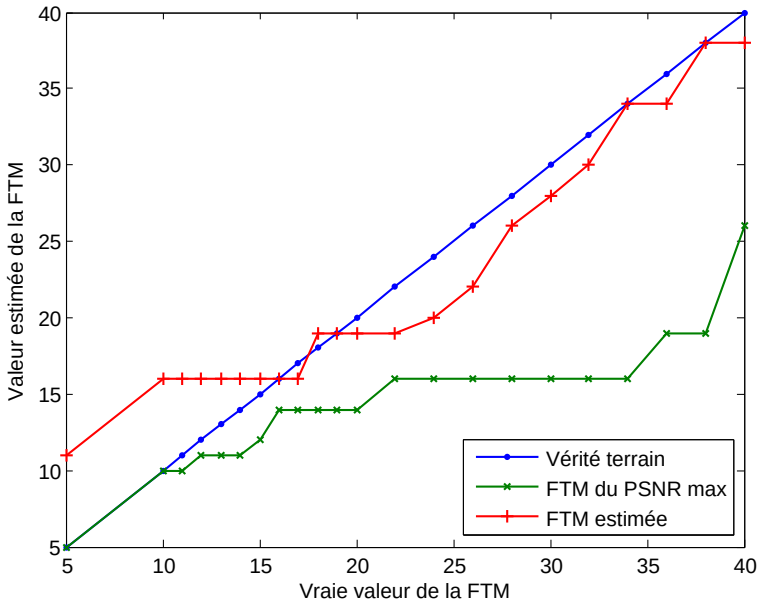



FIGURE 5.7 – Valeurs de FTM estimées pour *LosAngeles1* avec les seuils (1).

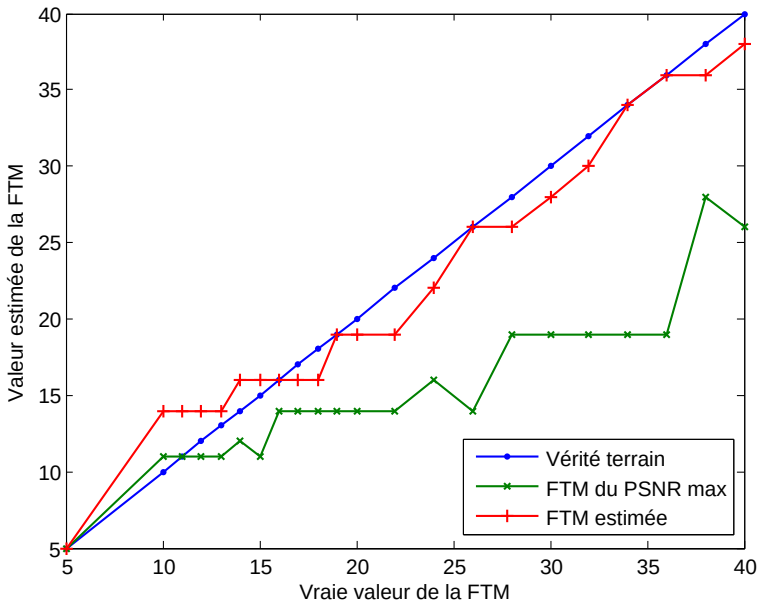



FIGURE 5.8 – Valeurs de FTM estimées pour *NewYork3* avec les seuils (1).

| Image test | Écart moyen de FTM | Écart min | Écart max |
|---|---|---|---|
| Hambourg1 | 2.18 | 0 | 7 |
| LosAngeles1 | 2.95 | 0 | 6 |
| NewYork3 | 2.18 | 0 | 5 |
| Toulouse1 | 1.91 | 0 | 5 |
| SalonProvence3 | 0.73 | 0 | 2 |
| Moyenne | 1.99 | 0 | 5.00 |

TABLE 5.2 – Estimation de la FTM avec les seuils (2) $(a_f; b_f) = (0.002; 0.04)$.

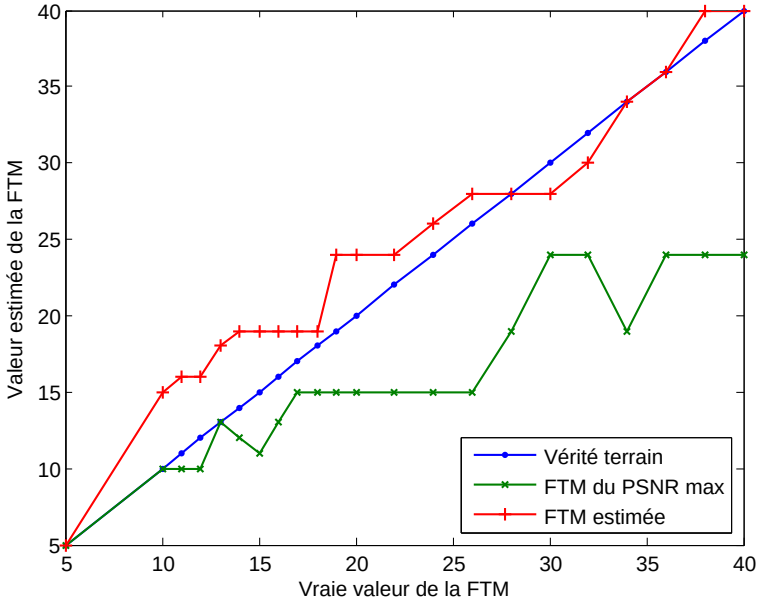



FIGURE 5.9 – Valeurs de FTM estimées pour *Toulouse1* avec les seuils (1).

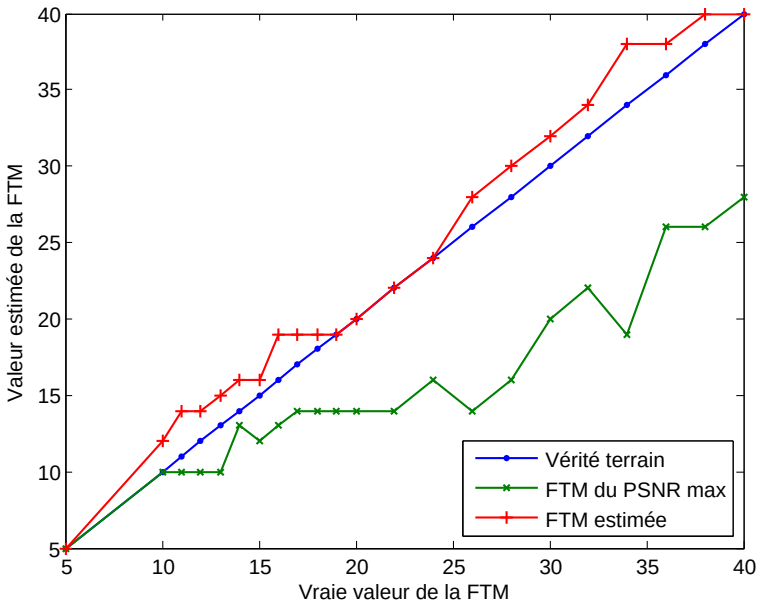



FIGURE 5.10 – Valeurs de FTM estimées pour *SalonProvence3* avec les seuils (1).

| Image test | Écart moyen de FTM | Écart min | Écart max |
|---|---|---|---|
| Hambourg1 | 2.32 | 0 | 6 |
| LosAngeles1 | 2.36 | 0 | 6 |
| NewYork3 | 1.36 | 0 | 3 |
| Toulouse1 | 2.82 | 0 | 5 |
| SalonProvence3 | 1.36 | 0 | 4 |
| Moyenne | 2.05 | 0 | 4.80 |

TABLE 5.3 – Estimation de la FTM avec les seuils (3) $(a_f; b_f) = (0.003; 0.04)$.

| Image test | Écart moyen de FTM | Écart min | Écart max |
|---|---|---|---|
| Hambourg1 | 2.18 | 0 | 7 |
| LosAngeles1 | 2.82 | 0 | 7 |
| NewYork3 | 2.05 | 0 | 5 |
| Toulouse1 | 1.64 | 0 | 5 |
| SalonProvence3 | 1.05 | 0 | 4 |
| Moyenne | 1.95 | 0 | 5.60 |

TABLE 5.4 – Estimation de la FTM avec les seuils (4) $(a_f; b_f) = (0.003; 0.03)$.

| Image test | Écart moyen de FTM | Écart min | Écart max |
|---|---|---|---|
| Hambourg1 | 1.86 | 0 | 5 |
| LosAngeles1 | 3.14 | 0 | 6 |
| NewYork3 | 2.09 | 0 | 4 |
| Toulouse1 | 2.45 | 0 | 5 |
| SalonProvence3 | 0.82 | 0 | 3 |
| Moyenne | 2.07 | 0 | 4.60 |

TABLE 5.5 – Estimation de la FTM avec les seuils (5) $(a_f; b_f) = (0.001; 0.05)$.

| Image test | Écart moyen de FTM | Écart min | Écart max |
|---|---|---|---|
| Hambourg1 | 1.86 | 0 | 5 |
| LosAngeles1 | 2.73 | 0 | 6 |
| NewYork3 | 1.41 | 0 | 4 |
| Toulouse1 | 2.86 | 0 | 6 |
| SalonProvence3 | 1.41 | 0 | 4 |
| Moyenne | 2.05 | 0 | 5.00 |

TABLE 5.6 – Estimation de la FTM avec les seuils (6) $(a_f; b_f) = (0.001; 0.06)$.

| Écart moyen de FTM | Écart min moyen | Écart max moyen | Écart min | Écart max |
|---|---|---|---|---|
| 2.01 | 0 | 4.97 | 0 | 7 |

TABLE 5.7 – Estimation de la FTM : moyennes sur les 6 seuils.

Cependant, le fait que la détermination des seuils soit effectuée sur les mêmes images que l'estimation de la FTM peut biaiser les résultats. Pour les prochaines expérimentations, nous distinguons donc, parmi les 5 images de test, les 3 images utilisées pour apprendre les seuils, des 2 autres images utilisées comme images de tests pour l'estimation de la FTM. De cette façon, la détermination des seuils et l'estimation de la FTM ne sont pas réalisées sur les mêmes images.

Étant donné que nous avons 5 images de test à notre disposition, il est possible de réaliser 10 expériences en combinant 3 images pour l'apprentissage des seuils et 2 pour

l'estimation de la FTM. Pour chaque expérience, on commence donc par déterminer les seuils à utiliser sur 3 images, puis l'algorithme de décision est appliqué avec ces seuils sur les 2 autres images de test, pour les 22 FTM disponibles. Les FTM estimées sont ensuite comparées aux véritables FTM des images (Tab. 5.8 - 5.17). Le Tableau 5.18 fournit un résumé des résultats de ces 10 expériences.

| Image test | Écart moyen de FTM | Écart min | Écart max |
|---|---|---|---|
| LosAngeles1 | 2.95 | 0 | 6 |
| NewYork3 | 2.18 | 0 | 5 |
| Moyenne | 2.57 | 0 | 5.50 |

TABLE 5.8 – Estimation de la FTM - expérimentation 1 : détermination des seuils sur *Hambourg1, Toulouse1 et SalonProvence3* (seuils (2) $(a_f; b_f) = (0.002; 0.04)$).

| Image test | Écart moyen de FTM | Écart min | Écart max |
|---|---|---|---|
| SalonProvence3 | 1.55 | 0 | 4 |
| NewYork3 | 1.45 | 0 | 4 |
| Moyenne | 1.50 | 0 | 4.00 |

TABLE 5.9 – Estimation de la FTM - expérimentation 2 : détermination des seuils sur *Hambourg1, Toulouse1 et LosAngeles1* (seuils (1) $(a_f; b_f) = (0.002; 0.05)$).

| Image test | Écart moyen de FTM | Écart min | Écart max |
|---|---|---|---|
| LosAngeles1 | 2.32 | 0 | 6 |
| SalonProvence3 | 1.55 | 0 | 4 |
| Moyenne | 1.93 | 0 | 5.00 |

TABLE 5.10 – Estimation de la FTM - expérimentation 3 : détermination des seuils sur *Hambourg1, Toulouse1 et NewYork3* (seuils (1) $(a_f; b_f) = (0.002; 0.05)$).

| Image test | Écart moyen de FTM | Écart min | Écart max |
|---|---|---|---|
| Toulouse1 | 2.45 | 0 | 5 |
| NewYork3 | 2.09 | 0 | 4 |
| Moyenne | 2.27 | 0 | 4.50 |

TABLE 5.11 – Estimation de la FTM - expérimentation 4 : détermination des seuils sur *Hambourg1, LosAngeles1 et SalonProvence3* (seuils (5) $(a_f; b_f) = (0.001; 0.05)$).

| Image test | Écart moyen de FTM | Écart min | Écart max |
|---|---|---|---|
| LosAngeles1 | 3.14 | 0 | 6 |
| Toulouse1 | 2.45 | 0 | 5 |
| Moyenne | 2.80 | 0 | 5.50 |

TABLE 5.12 – Estimation de la FTM - expérimentation 5 : détermination des seuils sur *Hambourg1, NewYork3 et SalonProvence3* (seuils (5) $(a_f; b_f) = (0.001; 0.05)$).

| Image test | Écart moyen de FTM | Écart min | Écart max |
|---|---|---|---|
| Toulouse1 | 2.50 | 0 | 5 |
| SalonProvence3 | 1.55 | 0 | 4 |
| Moyenne | 2.02 | 0 | 4.50 |

TABLE 5.13 – Estimation de la FTM - expérimentation 6 : détermination des seuils sur *Hambourg1, LosAngeles1 et NewYork3* (seuils (1) $(a_f; b_f) = (0.002; 0.05)$).

| Image test | Écart moyen de FTM | Écart min | Écart max |
|---|---|---|---|
| Hambourg1 | 2.18 | 0 | 7 |
| NewYork3 | 2.18 | 0 | 5 |
| Moyenne | 2.18 | 0 | 6.00 |

TABLE 5.14 – Estimation de la FTM - expérimentation 7 : détermination des seuils sur *LosAngeles1, Toulouse1 et SalonProvence3* (seuils (2) $(a_f; b_f) = (0.002; 0.04)$).

| Image test | Écart moyen de FTM | Écart min | Écart max |
|---|---|---|---|
| Hambourg1 | 2.18 | 0 | 7 |
| LosAngeles1 | 2.95 | 0 | 6 |
| Moyenne | 2.57 | 0 | 6.50 |

TABLE 5.15 – Estimation de la FTM - expérimentation 8 : détermination des seuils sur *NewYork3, Toulouse1 et SalonProvence3* (seuils (2) $(a_f; b_f) = (0.002; 0.04)$).

| Image test | Écart moyen de FTM | Écart min | Écart max |
|---|---|---|---|
| Hambourg1 | 2.32 | 0 | 6 |
| SalonProvence3 | 1.36 | 0 | 4 |
| Moyenne | 1.84 | 0 | 5.00 |

TABLE 5.16 – Estimation de la FTM - expérimentation 9 : détermination des seuils sur *LosAngeles1, Toulouse1 et NewYork3* (seuils (3) $(a_f; b_f) = (0.003; 0.04)$).

| Image test | Écart moyen de FTM | Écart min | Écart max |
|---|---|---|---|
| Hambourg1 | 2.32 | 0 | 6 |
| Toulouse1 | 2.82 | 0 | 5 |
| Moyenne | 2.57 | 0 | 5.50 |

TABLE 5.17 – Estimation de la FTM - expérimentation 10 : détermination des seuils sur *LosAngeles1, NewYork3 et SalonProvence3* (seuils (3) $(a_f; b_f) = (0.003; 0.04)$).

| Écart moyen de FTM | Écart min moyen | Écart max moyen | Écart min | Écart max |
|---|---|---|---|---|
| 2.23 | 0 | 5.20 | 0 | 7 |

TABLE 5.18 – Estimation de la FTM : moyenne sur les 10 expérimentations.

Ces dernières expérimentations correspondent à un cas d'application réaliste puisque les images de tests sont différentes des images de détermination des seuils et des images d'apprentissage. Ainsi, au vu des résultats, notre méthode d'estimation de FTM, basée apprentissage de dictionnaires et accompagnée d'un algorithme de décision et de détermination des seuils, semble efficace puisque l'écart moyen obtenu lors des dernières expérimentations entre les FTM estimées et les véritables FTM est de 2.23, alors que visuellement, la différence entre deux images présentant un écart de FTM de 3 est quasiment imperceptible (Fig. 5.11).

Dans le cas d'*Airbus Defence & Space*, des mires sont utilisées en dessous de deux mètres de GSD, la précision atteinte étant de l'ordre de 1 sur une plage d'utilisation de FTM entre 10 et 20. Au-delà de deux mètres de GSD, les mires deviennent trop petites et des patterns naturels sont utilisés, tels que des ponts. Pour des FTM entre 20 et 35, la précision obtenue est alors d'environ 5. La précision moyenne atteinte avec notre algorithme d'estimation (2.23), calculée pour des FTM entre 5 et 40 en ne considérant qu'une seule image test pour l'estimation, est ainsi meilleure que celle obtenue en utilisant des patterns naturels, sans avoir à rechercher de tels patterns dans l'image. Quant au cas utilisant une mire, bien que notre algorithme n'atteigne pas la même précision, il présente l'intérêt de se passer de mire.

(a) FTM=19 (b) FTM=22

FIGURE 5.11 – Portion de *New York* (250x250) pour deux valeurs de FTM proches : FTM=19 (a) et FTM=22 (b).

## 5.5 Conclusion

Dans ce chapitre, nous avons décrit une nouvelle méthode d'estimation de FTM à partir d'images quelconques prises par un satellite. Par rapport à la littérature, cette méthode ne nécessite pas de rechercher dans l'image de forts contours, de pointer le satellite vers une mire sur Terre, ou encore de travailler sur des images d'étoiles. A la différence des méthodes de la littérature cherchant à calculer la PSF ou la LSF de l'instrument pour en déduire la FTM, la méthode que nous avons présentée est basée sur l'apprentissage d'un dictionnaire structuré pour chaque valeur de FTM. La décomposition d'une image test de FTM inconnue sur chacun de ces dictionnaires, associée à un algorithme de décision, permet d'estimer une valeur de FTM. Les résultats obtenus semblent prometteurs.

Toutefois, certaines améliorations pourraient encore améliorer les résultats. En observant les résultats des estimations pour les seuils (1) par exemple (Fig. 5.6 - 5.10), on s'aperçoit que l'algorithme de décision permettant d'estimer la FTM est plus efficace que la méthode consistant à simplement sélectionner la FTM correspondant au PSNR maximal pour des FTM supérieures à 16 environ. En dessous, la FTM correspondant au PSNR maximal pourrait être choisie comme estimation, ce qui améliorerait les résultats pour l'estimation des FTM faibles, même si ces FTM correspondent à des images de faible qualité.

De plus, lors de l'étape de détermination des seuils, qui sont utiles à l'algorithme de décision, il pourrait être intéressant de chercher des fonctions de seuils plus complexes que de simples fonctions affines, choisies pour leur simplicité.

Enfin, chaque estimation de FTM a été réalisée dans ce chapitre en utilisant une seule image test avec une valeur de FTM précise. Puis des écarts moyens entre l'estimation de la FTM et sa vraie valeur ont été calculés pour 22 valeurs de FTM différentes allant de 5 à 40, afin d'étudier l'estimation sur une large plage de valeurs de FTM, et pour différentes images test. Ces expérimentations permettent donc de quantifier la précision de l'estimation qu'il est possible d'obtenir en utilisant une seule image test pour réaliser cette estimation, c'est-à-dire à partir d'une seule mesure, ce qui est une contrainte relativement forte. Dans le cas pratique où l'on souhaiterait estimer la FTM d'un système optique, la multiplication des images de test prises par ce système, et donc des mesures, devrait permettre d'améliorer encore la fiabilité de l'estimation obtenue.

# Chapitre 6

# Discrimination des dictionnaires pour la reconnaissance de scènes

Suite aux applications à la compression et à l'estimation de FTM des dictionnaires structurés appris, intéressons nous désormais à une dernière application : la reconnaissance de scènes. L'objectif est de reconnaître au sein d'une image les types de scènes présents, correspondant à des types de paysages, parmi un ensemble de scènes prédéfinies connues telles que de la ville, de la mer, du désert ou encore des nuages. Nous sommes ainsi confrontés à un problème de classification supervisée, où chaque type de scène correspond à une classe particulière. L'ensemble des classes est donc connu et nous souhaitons pouvoir segmenter une image test en plusieurs classes selon les scènes présentes dans l'image, tout en étant capable de les identifier et de façon automatique. Cette étude correspond à des cas pratiques d'analyse d'images satellites. Cela peut en effet être utile pour identifier une zone urbaine et en étudier la croissance. On peut également penser à un suivi de la déforestation, une fois la zone de forêt identifiée.

Dans un premier temps, nous décrirons la méthode permettant de réaliser une classification supervisée à l'aide de dictionnaires appris. Nous chercherons ensuite à ajouter un critère de discrimination à l'apprentissage des dictionnaires afin de les rendre plus efficaces dans le cadre de la classification. Puis nous présenterons les étapes de lissage des labels des classes obtenus, nécessaire à l'obtention d'une segmentation propre. Enfin, nous expérimenterons notre méthode sur des images de texture tout d'abord, afin de nous comparer à l'état de l'art sur une base d'images connue, puis sur des images satellites.

## 6.1 De la classification supervisée pour de la reconnaissance de scènes

Nous avons dans le premier chapitre présenté plusieurs articles traitant du problème de classification supervisée à l'aide de dictionnaires appris. Dans la lignée des travaux de *Mairal et al.* [MBP+08a] où un dictionnaire est appris par classe, nous apprenons une structure de dictionnaires par classe. Une Structure Adaptative est ainsi apprise

par classe, car nous pensons qu'elle a le pouvoir de davantage s'attacher aux données d'apprentissage qu'un dictionnaire "plat". Le but est ici de spécialiser le dictionnaire à une classe en particulier et la Structure Adaptative, de part le nombre de dictionnaires qu'elle contient, a cette capacité de spécialisation, comme nous avons pu le montrer dans le cadre de l'observation persistante.

De plus, comme montré dans [MBP+08a], ajouter un critère de discrimination lors de l'apprentissage des dictionnaires, afin de les rendre plus adaptés au problème de classification, permet d'améliorer les résultats. C'est pourquoi nous cherchons à apprendre des Structures Adaptatives discriminantes, spécialement apprises pour le problème de classification supervisée.

Une classification par pixel est alors réalisée en considérant un patch carré autour de chaque pixel. Chaque patch de l'image test à classifier est décomposé sur les différentes structures pour une certaine parcimonie $L$, par une décomposition de type MP. Puis, les erreurs de reconstruction des patchs de l'image test sur les différents dictionnaires sont utilisées pour réaliser la classification de chaque pixel de l'image. Ainsi, pour un patch $y$, une erreur d'approximation est calculée pour chaque dictionnaire $D_c$, $c \in [1, .., C]$, avec $C$ le nombre de classes, de la façon suivante :

$$R(y, D_c) = ||y - D_c\, x||_2^2 \tag{6.1}$$

avec $x$ le vecteur de coefficients contenant $L$ valeurs non nulles. Nous normalisons alors le résidu de la décomposition obtenu par la norme du patch à classifier, afin de ne pas donner plus d'importance à un patch de norme importante lors de l'étape de lissage, et l'erreur devient :

$$Err(y, D_c) = \frac{R(y, D_c)}{||y||_2^2} \tag{6.2}$$

Une classification simple serait ensuite de chercher pour chaque patch $y$ la classe $\hat{c}$ minimisant l'erreur de reconstruction $Err(y, D_c)$ :

$$\hat{c} = \underset{c=1,...,C}{\arg\min}\, Err(y, D_c) \tag{6.3}$$

Mais cela peut conduire à une segmentation très fractionnée, c'est-à-dire composée de nombreux petits labels. C'est pourquoi il est nécessaire d'appliquer à la place une méthode de lissage à partir des données $Err(y, D_c)$.

## 6.2 Apprentissage de dictionnaires discriminants

### 6.2.1 Principe

L'intérêt d'apprendre des dictionnaires discriminants, et non seulement pour de la reconstruction, est de les rendre davantage adaptés au problème de classification. L'objectif est alors que chaque dictionnaire d'une classe $c$ soit efficace pour représenter les données de sa classe, mais également peu performant pour représenter les données des autres classes. Les dictionnaires sont donc appris en considérant les données de leur

propre classe ainsi que celles des autres classes. Au sein des Structures Adaptatives, les dictionnaires sont rendus discriminants au premier niveau, où il n'existe qu'un unique dictionnaire par structure. Les niveaux suivants sont appris comme précédemment sur les résidus du niveau précédent avec K-SVD [AEB06] pour une parcimonie de 1 atome. La discrimination est simplement réalisée au premier niveau afin de limiter la complexité et donc le temps d'apprentissage des dictionnaires. Mais les niveaux suivants dépendant du premier niveau, tous les niveaux sont impactés par la discrimination du premier niveau.

### 6.2.2 Modèle discriminant

#### Modèle initial

Comme pour K-SVD, les dictionnaires sont appris en mettant à jour tour à tour chaque atome. Cela permet d'exprimer une fonction de coût simple à optimiser pour chaque atome. Afin d'ajouter de la discrimination aux dictionnaires du premier niveau des différentes Structures Adaptatives, la fonction de coût à minimiser afin de mettre à jour chaque atome est modifiée. Dans K-SVD, l'atome permettant de représenter au mieux un certain nombre de vecteurs de données est appris. Ici, l'atome appris doit comme pour K-SVD représenter certaines données de sa propre classe le mieux possible, mais doit également représenter les données des autres classes avec le plus d'erreur possible.

Le problème d'optimisation à résoudre pour apprendre chaque atome $d$ du dictionnaire $D_c$ de la classe $c$ peut donc s'écrire ainsi :

$$\min_{d \in \mathbb{R}^n} [||Y_c^R - dd^T Y_c^R||_F^2 - \lambda ||Y_{c_{\neq}} - dd^T Y_{c_{\neq}}||_F^2] \; avec \; ||d||_2 = 1 \tag{6.4}$$

La fonction de coût est composée de deux termes : le premier est le terme de reconstruction cherchant à adapter l'atome à une partie des données de la classe $c$, et le second est le terme de discrimination cherchant à l'éloigner des données des autres classes. En minimisant cette fonction de coût, on cherche l'atome minimisant la représentation d'une partie des données appartenant à la classe de l'atome $Y_c^R$ et maximisant la représentation des données des autres classes $Y_{c_{\neq}}$. La restriction $R$ sur $Y_c^R$ correspond à la même restriction utilisée au sein de K-SVD, c'est-à-dire limitant les données dans $Y_c$ à celles utilisant l'atome $d$ dans leur représentation.

Le paramètre $\lambda$, positif, permet d'équilibrer les deux termes, par le rapport des cardinaux des deux ensembles $Y_c^R$ et $Y_{c_{\neq}}$, et de régler le compromis entre reconstruction et discrimination, à l'aide d'un paramètre $\alpha$ positif constant :

$$\lambda = \frac{\#Y_c^R}{\#Y_{c_{\neq}}} \times \alpha \tag{6.5}$$

avec $\#Y_c^R$ correspondant au nombre de vecteurs de données de $Y_c^R$.

**Minimisation de la fonction de coût**

La fonction de coût à minimiser peut également s'écrire :

$$\min_{d\in\mathbb{R}^n}[tr((Y_c^R - dd^TY_c^R)^T(Y_c^R - dd^TY_c^R)) - \lambda tr((Y_{c_\neq} - dd^TY_{c_\neq})^T(Y_{c_\neq} - dd^TY_{c_\neq}))] \quad (6.6)$$

Cela est équivalent à :

$$\begin{aligned}\min_{d\in\mathbb{R}^n}[tr({Y_c^R}^TY_c^R - {Y_c^R}^Tdd^TY_c^R - {Y_c^R}^Tdd^TY_c^R + {Y_c^R}^Tdd^Tdd^TY_c^R) \\ - \lambda tr(Y_{c_\neq}^TY_{c_\neq} - Y_{c_\neq}^Tdd^TY_{c_\neq} - Y_{c_\neq}^Tdd^TY_{c_\neq} + Y_{c_\neq}^Tdd^Tdd^TY_{c_\neq})]\end{aligned} \quad (6.7)$$

D'après la contrainte $||d||_2 = 1$, on a $d^Td = 1$. On simplifie donc la fonction de coût à minimiser :

$$\min_{d\in\mathbb{R}^n}[tr({Y_c^R}^TY_c^R - {Y_c^R}^Tdd^TY_c^R) - \lambda tr(Y_{c_\neq}^TY_{c_\neq} - Y_{c_\neq}^Tdd^TY_{c_\neq})] \quad (6.8)$$

Afin de résoudre ce problème de minimisation avec la contrainte $d^Td = 1$, on applique ensuite la méthode des multiplicateurs de Lagrange pour minimiser la fonction $L(d, \mu)$ suivante :

$$L(d,\mu) = tr({Y_c^R}^TY_c^R - {Y_c^R}^Tdd^TY_c^R) - \lambda tr(Y_{c_\neq}^TY_{c_\neq} - Y_{c_\neq}^Tdd^TY_{c_\neq}) + \mu(d^Td - 1) \quad (6.9)$$

On a d'abord :

$$\frac{\partial L}{\partial \mu} = 0 \Leftrightarrow d^Td = 1 \quad (6.10)$$

Puis :

$$\frac{\partial L}{\partial d} = 0 \quad (6.11)$$

Donc :

$$\frac{\partial L}{\partial d} = \frac{\partial}{\partial d}tr(-{Y_c^R}^Tdd^TY_c^R) - \lambda\frac{\partial}{\partial d}tr(-Y_{c_\neq}^Tdd^TY_{c_\neq}) + 2\mu d^T = 0 \quad (6.12)$$

$$\frac{\partial L}{\partial d} = -2d^TY_c^R{Y_c^R}^T + 2\lambda d^TY_{c_\neq}Y_{c_\neq}^T + 2\mu d^T = 0 \quad (6.13)$$

$$\frac{\partial L}{\partial d} = d^T(-2Y_c^R{Y_c^R}^T + 2\lambda Y_{c_\neq}Y_{c_\neq}^T + 2\mu I) = 0 \quad (6.14)$$

On obtient ainsi l'équation suivante :

$$d^T(\lambda Y_{c_\neq}Y_{c_\neq}^T - Y_c^R{Y_c^R}^T) = -\mu d^T \quad (6.15)$$

Et en appliquant la transposée :

$$(Y_c^R{Y_c^R}^T - \lambda Y_{c_\neq}Y_{c_\neq}^T)d = \mu d \quad (6.16)$$

Cette équation est de la forme :

$$Ad = \mu d \quad (6.17)$$

avec

$$A = Y_c^R Y_c^{R^T} - \lambda Y_{c_{\neq}} Y_{c_{\neq}}^T \tag{6.18}$$

L'atome $d$ est donc un vecteur propre de A. On remarque qu'en choisissant $\lambda = 0$, on retrouve la mise à jour du K-SVD calculant une SVD d'une matrice d'erreur $E^R$ et choisissant le premier vecteur singulier de gauche, ce qui revient à choisir le vecteur propre de $E^R E^{R^T}$ correspondant à la valeur propre maximale. On applique ici la même procédure en mettant à jour $d$ avec le vecteur propre de $A$ correspondant à la valeur propre maximale.

**Une matrice d'affinité pour le terme de discrimination**

Afin d'adapter le terme de discrimination en fonction des classes, et même de l'atome à optimiser, une restriction sur le choix des données des autres classes est appliquée. Plutôt que de discriminer par rapport à l'ensemble des données des autres classes, on cherche à discriminer plus particulièrement par rapport aux classes les plus proches de la classe courante. Pour cela, une matrice d'affinité est calculée entre les différentes classes.

Pour chaque classe, on calcule un vecteur représentatif de façon similaire à l'apprentissage d'un atome dans l'algorithme K-SVD. Ainsi, chaque vecteur représentatif de classe est calculé sur les vecteurs de donnée de sa propre classe, $Y_c$, comme le vecteur propre de $Y_c Y_c^T$ associé à la plus grande valeur propre. A noter que la moyenne de chaque vecteur de $Y_c$ a au préalable été retirée (uniquement pour le calcul de la matrice d'affinité et non pour les apprentissages et décompositions) afin de ne pas trouver un vecteur constant comme vecteur représentatif.

Puis la valeur absolue du produit scalaire entre les différents vecteurs représentatifs des classes est calculée afin d'obtenir une matrice d'affinité carrée $C \times C$ (avec $C$ le nombre de classes). La diagonale comporte des 1 et le reste de la matrice des valeurs entre 0 et 1, selon les affinités entre les classes.

Cette matrice d'affinité permet de restreindre le nombre de vecteurs de $Y_{c_{\neq}}$ sélectionnés. En effet, au lieu de considérer l'ensemble des vecteurs d'apprentissage des autres classes ($Y_{c_{\neq}}$) que la classe courante $c$, on sélectionne dans ces classes un nombre de vecteurs plus ou moins important selon la valeur de la matrice d'affinité entre la classe $c$ et chacune des autres classes pour former la matrice $Y_{c_{\neq}}^{R_{AM}}$. Les valeurs de la matrice d'affinité, multipliées par cent, nous donnent ainsi les pourcentages des vecteurs d'apprentissage sélectionnés dans les classes autres que la classe courante $c$, puis concaténés dans $Y_{c_{\neq}}^{R_{AM}}$. Ce sont alors les vecteurs d'apprentissage les plus corrélés à $d$ qui sont choisis.

De cette façon, chaque dictionnaire est davantage discriminé par rapport aux classes dont sa classe est la plus proche, au lieu de le discriminer de la même manière par rapport à toutes les autres classes. En effet, deux classes déjà éloignées n'ont pas forcément besoin d'un critère de discrimination supplémentaire pour pouvoir être distinguées.

Suite à cette modification, la fonction de coût à minimiser devient donc :

$$\min_{d \in \mathbb{R}^n} [||Y_c^R - dd^T Y_c^R||_F^2 - \lambda ||Y_{c_{\neq}}^{R_{AM}} - dd^T Y_{c_{\neq}}^{R_{AM}}||_F^2] \; avec \; ||d||_2 = 1 \tag{6.19}$$

où :

$$\lambda = \frac{\#Y_c^R}{\#Y_{c_{\neq}}^{R_{AM}}} \times \alpha \tag{6.20}$$

et $R_{AM}$ la restriction sur $Y_{c_{\neq}}$ due à la matrice d'affinité. La matrice $A$ devient alors :

$$A = Y_c^R {Y_c^R}^T - \lambda Y_{c_{\neq}}^{R_{AM}} {Y_{c_{\neq}}^{R_{AM}}}^T \tag{6.21}$$

### 6.2.3 Algorithme d'apprentissage

Nous allons maintenant décrire l'algorithme permettant d'apprendre un dictionnaire discriminant par classe, ces dictionnaires constituant les dictionnaires au premier niveau des Structures Adaptatives.

Ces dictionnaires sont d'abord initialisés par des vecteurs d'apprentissage de leur propre classe, sélectionnés de manière aléatoire puis normalisés (afin d'être de normes unitaires).

L'algorithme itère entre une étape de décomposition des données de chacune des classes sur leur dictionnaire correspondant avec une parcimonie de 1 atome, et une étape de mise à jour des différents dictionnaires, atome après atome.

L'étape de décomposition est réalisée pour chacune des classes. On cherche alors pour chaque vecteur d'apprentissage de la classe courante $c$, l'atome du dictionnaire $D_c$ qui lui est le plus corrélé suivant l'algorithme de MP. Cela nous permettra de construire pour chaque atome $d$ la matrice $Y_c^R$ constituée des vecteurs d'apprentissage de la classe $c$ ($Y_c$) ayant choisi l'atome $d$ de $D_c$ à cette étape de décomposition.

Puis l'étape de mise à jour des dictionnaires est réalisée pour chaque classe $c$, atome après atome de $D_c$. Pour chaque atome, il est nécessaire de calculer la matrice $Y_{c_{\neq}}^{R_{AM}}$, constituée de vecteurs d'apprentissage des autres classes, leur nombre dépendant des valeurs de la matrice d'affinité (comme décrit précédemment), ainsi que la matrice $Y_c^R$. La matrice $A$ peut alors être calculée et l'atome $d$ est mis à jour par le vecteur propre de $A$ (normalisé) correspondant à la valeur propre maximale.

## 6.3 Étapes de lissage

Une méthode de classification simple consisterait à choisir pour chaque pixel le label correspondant au dictionnaire donnant l'erreur de reconstruction minimale du patch autour de ce pixel. Mais cela conduit généralement à une segmentation très bruitée de l'image test, avec de nombreux petits labels, comme nous le verrons lors des expérimentations. C'est pourquoi il est nécessaire d'appliquer une méthode de lissage afin de favoriser une segmentation de l'image en zones de tailles plus importantes.

Pour cela, différentes méthodes sont combinées. La première est une méthode d'expansion de labels par minimisation d'énergie sur un graph-cut. La seconde méthode, utilisée en complément de la première, est une méthode d'érosion permettant de supprimer les dernières petites zones indésirables.

### 6.3.1 Lissage par graph-cut

La première étape du lissage consiste à appliquer un lissage par graph-cut. Pour cela, un algorithme de minimisation d'une fonction d'énergie liée à un graph-cut est utilisé [BVZ01, KZ04, BK04], avec l'enveloppe Matlab [Bag06].

Un algorithme efficace cherchant une approximation de la solution à ce problème NP-difficile est présenté dans [BVZ01]. Il cherche à classifier l'image en attribuant un label $f_p$ à chaque pixel $p$ de façon à minimiser une fonction d'énergie de la forme :

$$E(f) = \sum_{p \in P} D_p(f_p) + \sum_{(p,q) \in N} V_{p,q}(f_p, f_q) \tag{6.22}$$

Le premier terme est le terme d'attache aux données. Il correspond à la somme sur tous les pixels $P$ du coût d'assignation d'un label $f_p$ à chaque pixel $p \in P$. Le second terme est le terme de lissage et est égal à la somme sur l'ensemble des pixels interagissants (typiquement adjacents) $N$ du coût associé à la paire de pixels voisins $p$ et $q$ de labels respectifs $f_p$ et $f_q$.

L'algorithme a ainsi besoin de deux matrices pour fonctionner : une matrice $Data_{cost}$ pour le coût lié aux données et une matrice $S_{cost}$ pour le coût lié au lissage. La matrice $Data_{cost}$ contient pour chaque pixel de l'image à classifier un coût associé à chaque classe (ou label de classe). Plutôt que d'appliquer un coût de 0 pour la classe correspondant au dictionnaire offrant la plus faible erreur de représentation, et de 1 pour les autres classes, comme cela est fait dans [MBP+08a], nous plaçons dans $Data_{cost}$ pour chaque pixel les $C$ valeurs $Err(y, D_c)$ (une valeur pour chaque classe $c$, avec $y$ le patch autour du pixel). Ainsi, les différentes classes sont hiérarchisées pour chaque pixel. La matrice $S_{cost}$ indique quant à elle les coûts associés à deux labels voisins. Si ces deux labels sont identiques, alors ce coût est nul. Dans le cas contraire, ce coût est constant quelque soit les labels. On se place ainsi dans le cas du modèle de Potts [Pot52] suivant :

$$E(f) = \sum_{p \in P} D_p(f_p) + \sum_{(p,q) \in N} u_{(p,q)}.T(f_p \neq f_q) \tag{6.23}$$

avec $T(f_p \neq f_q)$ égal à 1 si le label $f_p$ est différent du label $f_q$ et 0 sinon. Dans ce cas, les discontinuités entre toute paire de labels $(f_p, f_q)$ sont pénalisées de la même manière. Ce modèle encourage un étiquetage des labels en plusieurs régions dont les pixels dans chaque région partagent le même label. On devra alors jouer sur la valeur du paramètre $u_{(p,q)}$ afin de plus ou moins lisser la segmentation. En choisissant ce paramètre de lissage à 0, seul le coût lié aux données est pris en compte et la classification obtenue sélectionne donc pour chaque pixel la classe correspondant à l'erreur de représentation la plus faible. Au contraire, un paramètre de lissage trop important fusionnera de trop nombreux labels et la segmentation comportera moins de classes que souhaité.

Afin de minimiser la fonction d'énergie, une méthode d'expansion alpha des labels est appliquée sur le graph-cut [BVZ01]. Cette méthode modifie les labels des pixels à chaque étape pour diminuer la fonction d'énergie $E(f)$. Pour cela, à chaque itération, chaque label est étendu tour à tour, en cherchant l'expansion optimale pour chaque label. L'expansion peut alors modifier le label de nombreux pixels simultanément en les affectant au label courant, dit label alpha.

### 6.3.2 Érosion

Suite au lissage par graph-cut, une étape d'érosion [SE14] est appliquée sur les labels obtenus (et avec la même matrice de données $Data_{cost}$) afin de supprimer les petites zones restantes présentant un certain label et présentes au sein de plus grandes zones d'un autre label, sur les bords entre deux labels ou bien sur les bords de l'image.

Cet algorithme d'érosion alpha [SE14] (disponible sur le site du premier auteur) propose une variante de la fonction d'énergie 6.23 en accordant pour le lissage un poids moins important aux pixels voisins diagonaux qu'aux pixels voisins sur la même ligne ou la même colonne. L'algorithme cherche à éroder les petits segments (un segment correspondant à un ensemble de pixels connectés) en priorité afin de réduire la fonction d'énergie. Les segments de taille inférieure à une taille limite sont ainsi traités tour à tour, en commençant par le plus petit. Pour chaque segment, les labels des pixels sont modifiés un à un, érodés par les segments autour du segment courant. Si le nouvel étiquetage du segment diminue la fonction d'énergie, alors l'érosion du segment est acceptée, sinon elle est annulée.

Les auteurs dans [SE14] utilisent des dictionnaires appris avec l'algorithme RLS-DLA [SE10] pour classifier la même base d'images de textures que dans [MBP+08a]. Leur méthode de lissage utilise d'abord un filtrage Gaussien passe-bas, puis applique l'algorithme d'érosion alpha, suivi par quelques érosions supplémentaires sur les bords des labels afin de les lisser.

## 6.4 Expérimentations

Nous allons désormais tester notre méthode de classification et la comparer à l'état de l'art sur des images de texture communément utilisées dans le cadre de la classification supervisée. Puis nous traiterons le cas réel de la reconnaissance de scènes sur des images satellites. Mais d'abord, nous allons détailler quelques points de l'implémentation de l'algorithme de classification.

### 6.4.1 Pré-traitement des données

Afin de faciliter la classification, quelques pré-traitements, utilisés dans [MBP+08a], sont effectués sur les patchs d'apprentissage et de test.

Un masque Gaussien (d'écart-type 4) est tout d'abord appliqué via une multiplication de chaque pixel du patch par sa valeur correspondante au sein du masque. Le but est de donner d'avantage de poids aux pixels centraux du patch, les pixels périphériques pouvant se retrouver sur une autre classe que le pixel central lorsque les bords des classes sont traités. Le poids associé au pixel central est ainsi de 1 et diminue en s'en éloignant.

Chaque patch est également accentué à l'aide d'un filtre Laplacien (de taille 3x3 pixels), le patch filtré étant soustrait au patch original afin d'obtenir le patch accentué.

A noter cependant que pour le calcul de la matrice d'affinité, réalisé sur les données d'apprentissage, ces pré-traitements ne sont pas appliqués car nous avons pu observer qu'ils perturbaient le calcul des affinités entre classes.

### 6.4.2 Classification par pixel

Nous réalisons une classification par pixel, c'est-à-dire qu'un label est associé à chaque pixel, dans le but de rendre la classification plus précise, notamment au niveau des bords, par rapport à une classification par bloc. Ainsi, chaque pixel est traité en considérant un patch carré autour de ce pixel, et c'est ce patch qui est décomposé sur les différents dictionnaires afin de donner un label au pixel central.

Pour pouvoir également traiter les pixels présents sur les bords de l'image test, dont les patchs dépassent du cadre de l'image, des bords ont été ajoutés. Pour cela, un simple effet miroir des valeurs proches des bords de l'image a été réalisé.

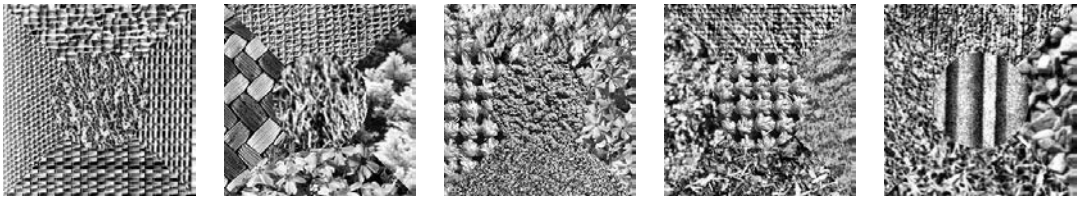
### 6.4.3 Expérimentations sur une base d'images de textures

Les premières expérimentations sont menées sur une base d'images de textures. Cette base a d'abord été introduite dans [RH99], et a depuis été utilisée dans plusieurs articles traitant de segmentation et classification de textures. Elle a été constituée à partir de textures provenant de l'album de Brodatz [Bro66], de la base de donnée *Vision Texture* du MIT et de la base de données d'images de textures *MeasTex*. Les images ont donc été capturées par différents équipements et sous différentes conditions. Elle est composée de 12 images test (Fig. 6.1), comportant chacune entre 2 et 16 textures différentes, et pour chaque image test, des images d'apprentissage correspondant aux différentes textures présentes. Les images de test et d'apprentissage ont été réalisées à partir de portions différentes de chaque texture. Elle est disponible en ligne [Ran].

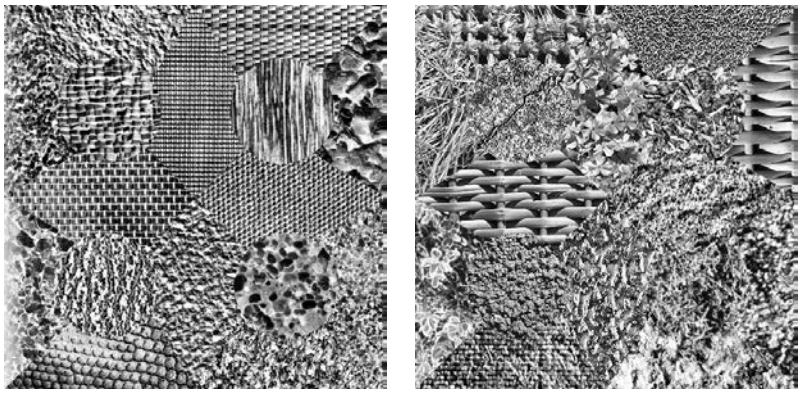
#### Paramètres

Pour chaque image test, une Structure Adaptative est apprise par classe qu'elle contient, sur les images d'apprentissage correspondantes. Des blocs de 8x8 pixels se chevauchant sont extraits de ces images 256x256 afin d'apprendre chaque structure sur 62001 blocs d'entraînement. Les dictionnaires au sein des structures sont complets et comportent donc 64 atomes. Le nombre d'atome est limité dans ce cadre de classification afin que chaque dictionnaire ne soit pas trop efficace en représentation et alors capable de représenter l'ensemble des classes, mais se limite à apprendre les caractéristiques principales de sa propre classe. Le premier niveau, comportant la discrimination, est appris en 50 itérations, tandis que les niveaux suivants sont appris en 10 itérations. Les dictionnaires sont initialisés sur des vecteurs d'entraînement sélectionnés aléatoirement. Le paramètre $\alpha$ (présent dans le calcul du $\lambda$) permettant de fixer la pondération entre reconstruction et discrimination au premier niveau des structures est fixé à $\frac{1}{40}$.

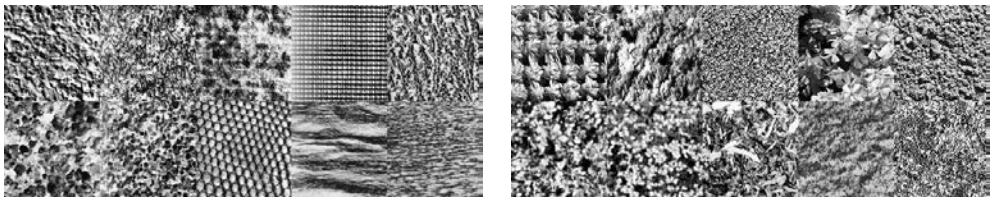
Les patchs de chaque image test, de taille 8x8 également, sont décomposés sur les dictionnaires des classes présentes dans l'image, pour une valeur de parcimonie de 2 atomes. En effet, nous ne cherchons pas ici à obtenir la meilleure approximation de chaque patch comme cela était le cas pour l'application de codage, mais à discriminer chaque patch selon son erreur d'approximation sur les différents dictionnaires. C'est pourquoi il est préférable de limiter la valeur de parcimonie à un nombre d'atomes



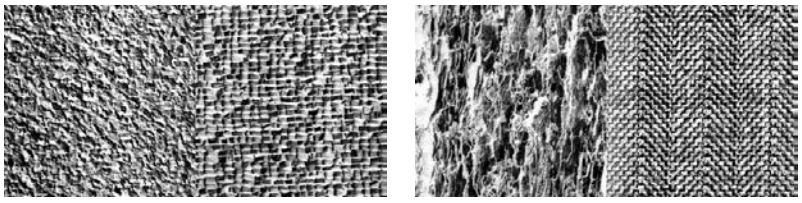
(a) Image test 1 (b) Image test 2 (c) Image test 3 (d) Image test 4 (e) Image test 5



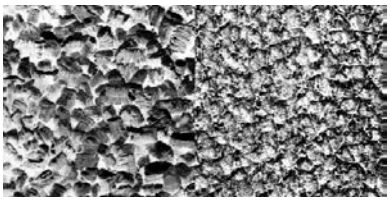
(f) Image test 6 (g) Image test 7



(h) Image test 8 (i) Image test 9



(j) Image test 10 (k) Image test 11



(l) Image test 12

FIGURE 6.1 – Base d'images de textures : images test.

relativement faible.

Concernant le lissage par graph-cut, le paramètre $u_{(p,q)}$ permettant de plus ou moins lisser la segmentation est fixé à une valeur de 0.16. Enfin, l'érosion finale permettant de nettoyer l'image segmentée est réalisée pour $\lambda = 2$ (pondération du lissage dans [SE14]). Les zones de moins de 2000 pixels sont obligatoirement érodées tandis que celles de plus de 10000 pixels ne le sont jamais. Entre ces deux limites, l'érosion dépend de la minimisation de la fonction de coût. Une légère érosion de 2 pixels est également réalisée sur les bords.

L'ensemble de ces paramètres a été fixé de manière empirique.

**Résultats**

L'étape de lissage par graph-cut contenant de l'aléatoire pour l'ordre d'expansion des labels à chaque itération, les résultats obtenus peuvent varier d'un lissage à l'autre, même si les mêmes dictionnaires et les mêmes paramètres sont utilisés. C'est pourquoi les deux étapes de lissage ont été réalisées 20 fois de façon indépendante. La Table 6.1 présente les résultats obtenus sur les 12 images tests. Y sont présentes pour chaque image l'erreur de classification/segmentation moyenne calculée sur les 20 essais, ainsi que l'erreur minimale et l'erreur maximale obtenues. On peut voir que les résultats varient finalement assez peu malgré l'aléa introduit dans le lissage.

Les erreurs de classification moyennes obtenues par notre algorithme sont ensuite comparées à celles obtenues par différentes méthodes de l'état de l'art (Table 6.2). L'article [RH99] introduisant la base de texture est tout d'abord présent dans la comparaison. De nombreuses méthodes de filtrage y sont comparées et le meilleur résultat obtenu pour chaque image est présenté. Puis, les auteurs de [MPO00] ont amélioré les résultats précédents en utilisant l'opérateur LBP (*Local Binary Pattern*) sur des patchs de texture et en en calculant l'histogramme pour caractériser une texture. Une version multi-échelle considérant plusieurs tailles de patchs y est également étudiée. Les auteurs de [DLMS07] ont ensuite proposé d'extraire des caractéristiques discriminantes de texture dans le domaine fréquentiel en appliquant une transformée de Fourier en coordonnées polaires, suivie par une réduction de dimension réalisée par une PCA ou le calcul de coefficients de Fisher. Des centroïdes sont ensuite calculés par classe par une méthode de quantification vectorielle. Les résultats de [MBP+08a] sont bien sûr présents dans la comparaison, en utilisant des dictionnaires pour la reconstruction (R) ou discriminants (D), et un lissage par graph-cut également. Enfin, les résultats récemment obtenus par l'érosion-$\alpha$ [SE14] sont ajoutés. Notre méthode permet d'obtenir les meilleurs résultats sur trois images et des résultats proches des meilleurs résultats (moins de 1% d'écart) sur les autres images, hormis deux images plus problématiques (les images 5 et 6). En moyenne, sur les douze images de test, le taux d'erreur de classification atteint 2.86%, soit une valeur semblable à celle présentée dans [SE14], et plus faible que les autres méthodes de l'état de l'art.

La Figure 6.2 montre l'image de labels obtenue après chaque étape de l'algorithme de classification. On remarque l'intérêt des méthodes de lissage étant donné qu'une simple classification par l'erreur de reconstruction minimale (Fig.6.2(b)) donne une

| Image test | Erreur moyenne | Erreur min | Erreur max |
|---|---|---|---|
| 1 | 1.25 | 1.22 | 1.27 |
| 2 | 3.42 | 3.38 | 3.45 |
| 3 | 3.05 | 3.05 | 3.05 |
| 4 | 2.59 | 2.55 | 2.65 |
| 5 | 6.60 | 6.59 | 6.62 |
| 6 | 8.20 | 8.04 | 8.24 |
| 7 | 2.36 | 2.34 | 2.38 |
| 8 | 3.13 | 2.97 | 3.31 |
| 9 | 2.06 | 2.03 | 2.11 |
| 10 | 0.23 | 0.23 | 0.23 |
| 11 | 0.43 | 0.43 | 0.43 |
| 12 | 0.94 | 0.94 | 0.94 |
| Moyenne | 2.86% | 2.81% | 2.89% |

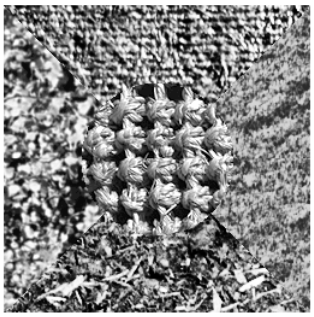
TABLE 6.1 – Erreurs de classification/segmentation (en %) sur les différentes images de test. Les étapes de lissage ont été lancées 20 fois et les résultats obtenus sont compris entre l'erreur min et l'erreur max, l'erreur moyenne étant la valeur moyenne sur les 20 essais.

| Image test | [RH99] | [MPO00] | [DLMS07] | [MBP$^+$08a](R) | [MBP$^+$08a](D) | [SE14] | Nous |
|---|---|---|---|---|---|---|---|
| 1 | 7.2 | 6.7 | 3.37 | 1.69 | 1.61 | 2.00 | **1.25** |
| 2 | 18.9 | 14.3 | 16.05 | 36.5 | 16.42 | **3.24** | 3.42 |
| 3 | 20.6 | 10.2 | 13.03 | 5.49 | 4.15 | 4.01 | **3.05** |
| 4 | 16.8 | 9.1 | 6.62 | 4.60 | 3.67 | **2.55** | 2.59 |
| 5 | 17.2 | 8.0 | 8.15 | 4.32 | 4.58 | **1.26** | 6.60 |
| 6 | 34.7 | 15.3 | 18.66 | 15.50 | 9.04 | **6.72** | 8.20 |
| 7 | 41.7 | 20.7 | 21.67 | 21.89 | 8.80 | 4.14 | **2.36** |
| 8 | 32.3 | 18.1 | 21.96 | 11.80 | **2.24** | 4.80 | 3.13 |
| 9 | 27.8 | 21.4 | 9.61 | 21.88 | **2.04** | 3.90 | 2.06 |
| 10 | 0.7 | 0.4 | 0.36 | **0.17** | **0.17** | 0.42 | 0.23 |
| 11 | **0.2** | 0.8 | 1.33 | 0.73 | 0.60 | 0.61 | 0.43 |
| 12 | 2.5 | 5.3 | 1.14 | **0.37** | 0.78 | 0.70 | 0.94 |
| Moyenne | 18.4% | 10.9% | 10.16% | 10.41% | 4.50% | 2.87% | **2.86%** |

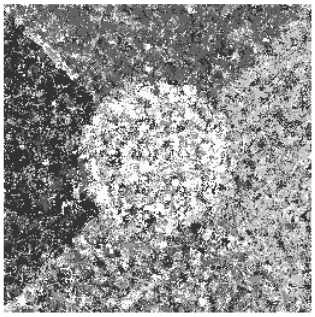
TABLE 6.2 – Erreurs de classification/segmentation (en %) sur les différentes images de test par rapport à différentes méthodes de l'état de l'art.

segmentation très bruitée. L'étape de lissage par graph-cut est alors primordiale pour créer des zones de labels plus grandes et supprimer la majorité des petites zones isolées (Fig. 6.2(c)). A noter que ce n'est pas l'image de labels Fig. 6.2(b) qui est donnée en entrée de l'algorithme de lissage par graph-cut, mais une erreur par pixel et par classe. L'étape finale d'érosion permet enfin de réaliser un dernier nettoyage de l'image de labels en érodant les dernières petites zones isolées et en appliquant une légère érosion

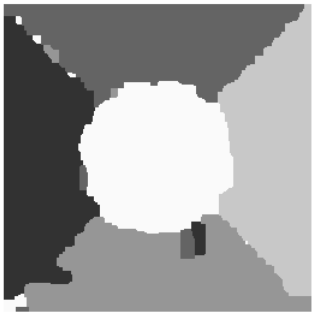
sur les bords (Fig. 6.2(d)). L'algorithme d'érosion prend en entrée la même matrice de coût d'attache aux données $Data_{cost}$ que l'algorithme de graph-cut, mais part de l'image de labels issue du lissage par graph-cut (Fig. 6.2(c)) comme segmentation initiale.



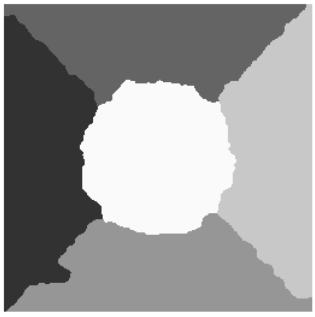
(a) Image test 4



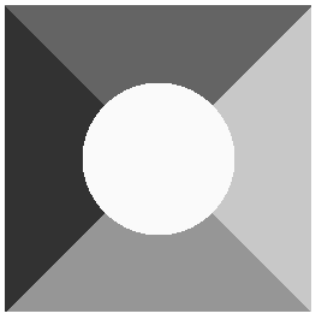
(b) Erreur min



(c) Après graph-cut



(d) Après érosion



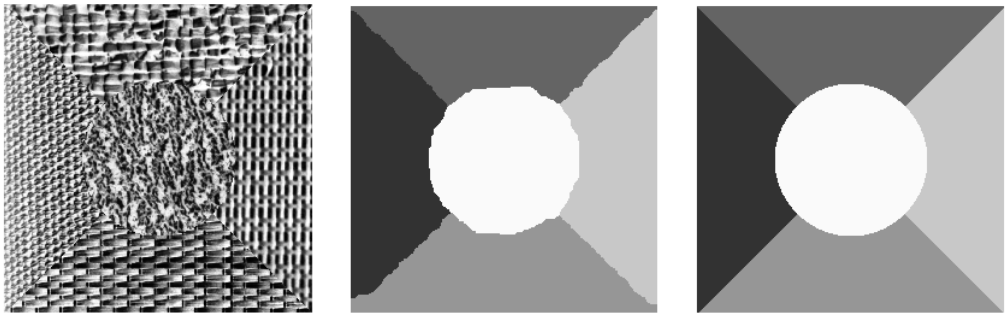
(e) Vérité terrain

FIGURE 6.2 – Image test 4 aux différentes étapes de l'algorithme de classification/segmentation.

Des résultats qualitatifs de segmentations sont présentés sur quelques images de test (Fig.6.3-6.6). Hormis quelques effets de bords entre deux classes différentes, l'algorithme permet d'obtenir une excellente classification sur ces images.

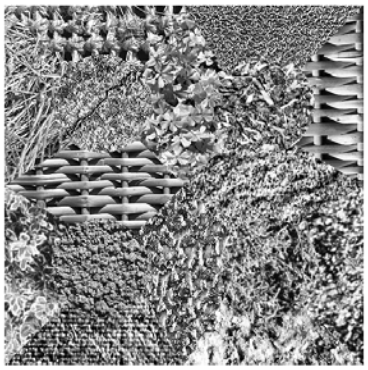
Cependant, deux images posent davantage de problèmes que les autres : les images 6 et 5, seules images au dessus des 4% d'erreurs de classification.

En observant l'image test 6 et les images de labels aux différentes étapes de l'algorithme de classification (Fig. 6.7), on observe sur l'image de labels finale (Fig. 6.7(d)) que les problèmes majeurs apparaissent sur les textures en bas à gauche (le label déborde) et en bas à droite (mauvais label sur l'ensemble de la texture). Sur l'image de labels issue du graph-cut (Fig. 6.7(c)), on remarque que les mauvaises classifications sur ces deux textures sont respectivement présentes sur la partie gauche de la première et la partie haute de la seconde. En regardant ces zones sur ces textures sur l'image originale (Fig. 6.7(a)), on s'aperçoit alors qu'elles semblent sur-exposées par rapport au reste de la texture, alors que cette sur-exposition n'est pas présente sur les images d'entraînement des classes correspondantes. Cela peut ainsi perturber l'algorithme de classification. On remarque d'ailleurs que les textures présentant des patterns réguliers

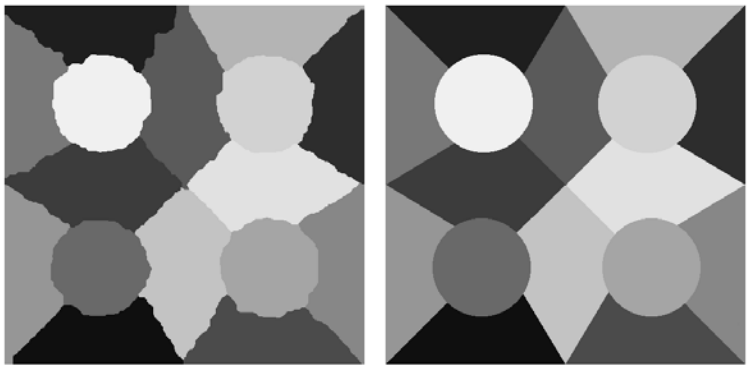


(a) Image test 1 (b) Image de segmentation (c) Vérité terrain

FIGURE 6.3 – Image test 1 et sa classification/segmentation par rapport à la vérité terrain.



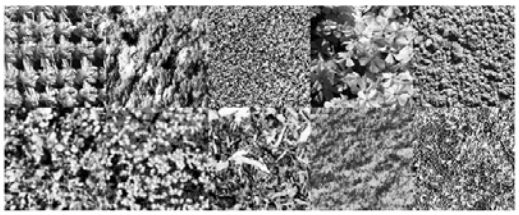
(a) Image test 7



(b) Image de segmentation (c) Vérité terrain

FIGURE 6.4 – Image test 7 et sa classification/segmentation par rapport à la vérité terrain.

et de petite taille sont facilement classifiées par l'algorithme, même avant les étapes de lissage (Fig. 6.7(b)).

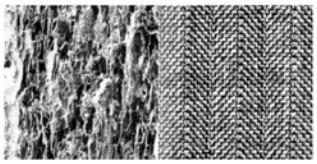


(a) Image test 9

(b) Image de segmentation

(c) Vérité terrain

FIGURE 6.5 – Image test 9 et sa classification/segmentation par rapport à la vérité terrain.



(a) Image test 11

(b) Image de segmentation

(c) Vérité terrain

FIGURE 6.6 – Image test 11 et sa classification/segmentation par rapport à la vérité terrain.

Afin de corriger le problème lié à la sur-exposition de portions de l'image de test, nous avons enrichi la base d'apprentissage en y ajoutant des versions sur-exposées des images d'apprentissage. Les images sur-exposées $I_{expo}$ sont créées à partir des images $I$ de la façon suivante :

$$I_{expo} = I + expo \tag{6.24}$$

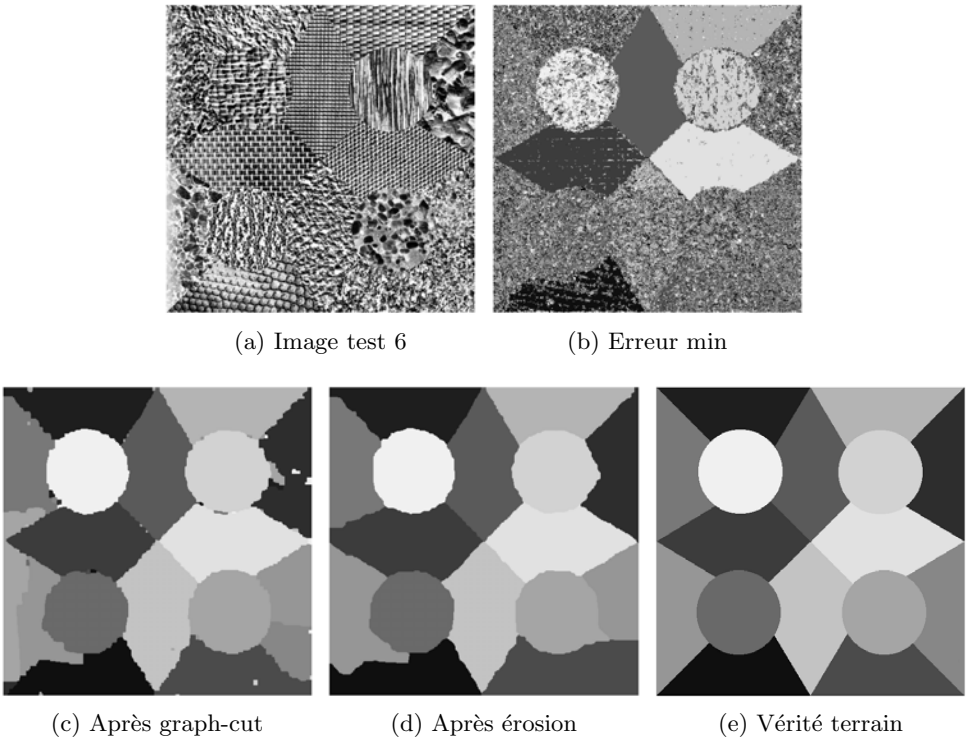
$$M = \max(I_{expo}) \tag{6.25}$$

$$m = \min(I) \tag{6.26}$$

$$I_{expo} = round((I_{expo} - m) \times \frac{255}{M - m}) \tag{6.27}$$

avec $expo$ la valeur de l'offset ajouté à l'image $I$, $M$ la valeur maximale de l'image $I_{expo}$ suite à l'ajout de l'offset d'exposition $expo$ à l'image $I$ et $m$ la valeur minimale de l'image $I$. La fonction $round$ représente la fonction d'arrondi à l'entier le plus proche.

Des images pour différents degrés de sur-expositions ont ainsi été créées pour enrichir la base d'apprentissage en utilisant plusieurs valeurs d'offset $expo$ (100, 300, 500 et



(a) Image test 6 (b) Erreur min

(c) Après graph-cut (d) Après érosion (e) Vérité terrain

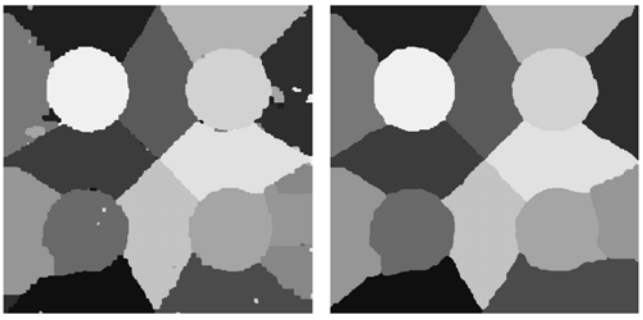
FIGURE 6.7 – Image test 6 aux différentes étapes de l'algorithme de classification/segmentation.

700). Nous avons choisi d'équilibrer le nombre de vecteurs d'apprentissage originaux et celui de vecteurs d'apprentissage sur-exposés. Ainsi, sur 80000 vecteurs d'apprentissage sélectionnés par classe, 40000 ne sont pas sur-exposés et 40000 le sont (10000 pour chaque valeur d'*expo*). Les dictionnaires ont donc été appris sur ces nouvelles données.

Les résultats obtenus varient alors entre 2 cas, selon l'aléa sur l'expansion des labels lors du lissage par graph-cut (Fig. 6.8). Le cas 1 est le cas favorable : les nouvelles données d'apprentissage ont permis d'apprendre des dictionnaires contenant davantage d'informations, ce qui permet de corriger le problème de sur-exposition et d'obtenir une erreur de classification de 2.35% sur la figure 6.8(b). Pour le cas 2, le problème est corrigé pour la texture en bas à gauche, mais persiste en bas à droite, le label de la texture étant entièrement faux, ce qui donne une erreur de 7.50% (Fig. 6.8(d)). En fait, en regardant l'image de labels après le lissage par graph-cut pour ce cas 2 (Fig. 6.8(c)), sur la texture en bas à droite, seule la partie sur-exposée au centre de la texture est mal classifiée, le reste de la texture étant assigné au bon label. C'est ensuite l'étape d'érosion qui assigne le mauvais label à l'ensemble de la texture. En répétant l'opération de lissage 20 fois de façon indépendante, on obtient des valeurs variant entre 2.17% et 7.65%, l'erreur moyenne étant de 4.36%, tandis qu'elle était de 8.20% sans enrichissement de la base. En prenant en compte cette amélioration sur la segmentation

de l'image 6, l'erreur de classification moyenne sur les 12 images tomberait à 2.54%. Cet enrichissement de la base d'apprentissage par sur-exposition des images qu'elle contient améliore donc les résultats, mais des problèmes persistent encore dans certains cas. Néanmoins, l'idée semble prometteuse et mériterait d'être creusée davantage en augmentant par exemple le nombre de niveaux de sur-expositions, voire en utilisant un modèle de sur-exposition plus complexe et plus réaliste.

(a) Après graph-cut (cas 1) (b) Après érosion (cas 1)



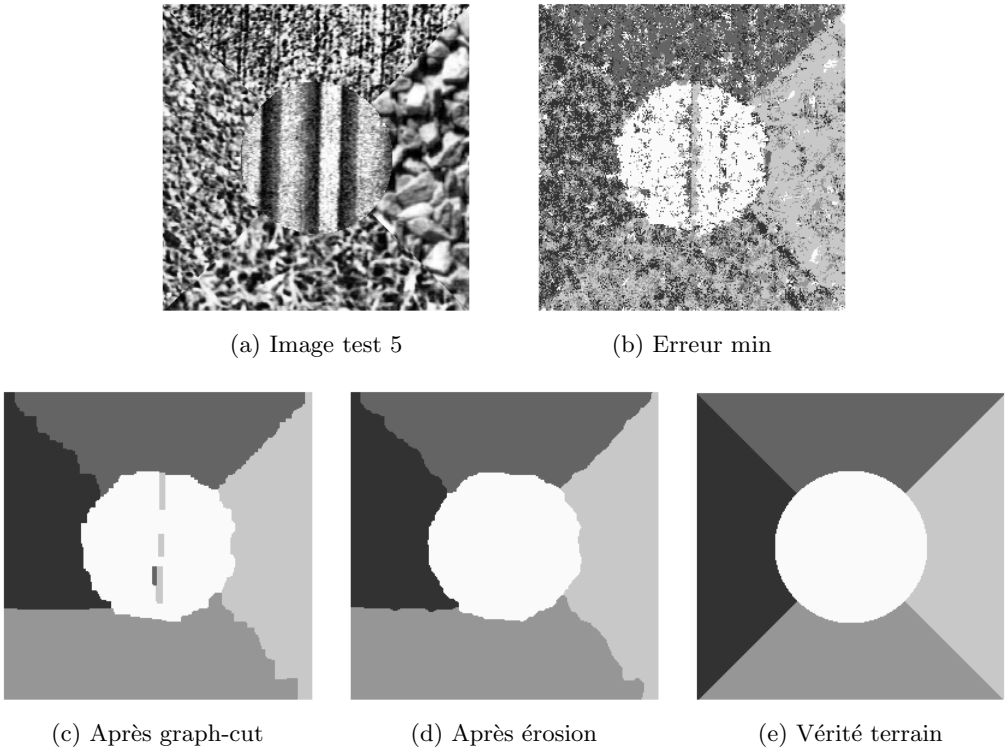
(c) Après graph-cut (cas 2) (d) Après érosion (cas 2)

FIGURE 6.8 – Images de segmentation de l'image test 6 après enrichissement de la base d'apprentissage par sur-exposition (cas 1 : 2.35% ; cas 2 : 7.50%).

Enfin, pour l'image 5 (Fig. 6.9), le problème est moins grave puisque les bons labels sont trouvés, mais le label de la texture du bas a tendance à trop s'étendre sur la texture de gauche. Cela est dû au fait que la frontière entre ces deux textures est difficilement perceptible, même à l'œil. La zone mal labellisée dans le coin en bas à gauche peut en effet facilement être assimilée au label de la texture du bas, même par un humain. Le fait que l'algorithme se trompe n'est donc pas surprenant. Augmenter la taille des blocs, ainsi que réduire le paramètre de lissage du graph-cut, pourraient peut-être aider dans cette situation.

### 6.4.4 Expérimentations sur des images satellites

Après avoir expérimenté l'algorithme de classification sur des images de textures, appliquons le désormais à des images satellites dans un cas réel de reconnaissance de



(a) Image test 5 (b) Erreur min

(c) Après graph-cut (d) Après érosion (e) Vérité terrain

FIGURE 6.9 – Image test 5 aux différentes étapes de l'algorithme de classification/segmentation.

scènes. Les classes considérées, indiquées par *Airbus Defence & Space*, sont au nombre de 7 et correspondent à du désert, de la ville, de la campagne, de la mer, de la neige, de la montagne et des nuages. Initialement, une classe forêt était considérée mais a finalement été écartée à cause du manque de données de forêt dans la base d'apprentissage disponible.

Les images sont initialement des images 12 bits sur quatre composantes (Rouge, Vert, Bleu et Infra Rouge), sur lesquelles un gain a été appliqué sur chaque bande afin de corriger les décalages de dynamiques entre les capteurs, ce qui augmente la dynamique des données.

**Paramètres**

Pour ces expérimentations, nous avons choisi d'augmenter la taille des blocs afin d'être capables d'apprendre des patterns de plus grande taille. Ainsi, les blocs d'apprentissage et de test sont de taille 13x13 pixels. Une Structure Adaptative est comme précédemment apprise par classe, sur 100000 vecteurs d'entraînement. Afin de réduire le temps d'apprentissage, plus important du fait de l'augmentation de la taille des blocs et du plus grand nombre de vecteurs d'apprentissage sélectionnés par classe, la taille de chaque dictionnaire au sein des structures est limitée à 100 atomes (les dictionnaires

sont donc sous-complets) et le premier niveau de chaque structure est appris en 25 itérations. La nature et la dynamique des données étant différentes, nous avons adapté le paramètre de lissage par graph-cut $u_{(p,q)}$ en le diminuant à une valeur de 0.03. Les valeurs des autres paramètres ont été conservées.

Les dictionnaires discriminants appris au premier niveau des Structures Adaptatives pour les différentes classes sur les images satellites sont représentés Figure 6.10.

**Ajout d'un terme de pénalité sur la couleur**

Afin de se concentrer sur l'apprentissage des structures et patterns spécifiques des classes grâce aux dictionnaires, l'apprentissage des dictionnaires et les décompositions des images de test sont réalisés sur les images de luminance des images satellites. Ces versions en niveau de gris des images d'apprentissage et de test sont obtenues via une combinaison linéaire des composantes Rouge, Vert et Bleu.

Mais étant donné que nous disposons également d'informations de couleurs via les quatre bandes Rouge, Vert, Bleu et Infra Rouge, nous décidons d'utiliser ces informations afin de créer un terme de pénalité sur la couleur pour ensuite le pondérer à celui lié aux dictionnaires, avant d'appliquer les étapes de lissage. Cela doit permettre d'améliorer la classification finale.

Pour chaque classe et chaque composante couleur (R, V, B et IR) un histogramme est calculé, sur 16 niveaux de quantification, sur les pixels des images d'apprentissage. Chaque histogramme est alors transformé en densité de probabilité en divisant chaque valeur de l'histogramme par la somme des valeurs de l'histogramme. Afin d'associer à chaque pixel une pénalité liée à la couleur, une densité de probabilité est calculée de la même façon pour chaque pixel sur le patch centré sur ce pixel, pour chaque composante couleur.

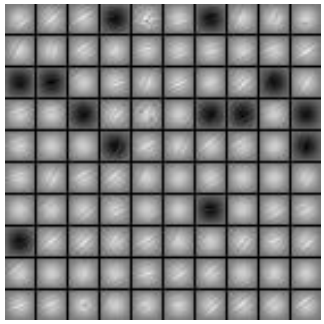
Nous utilisons ensuite une distance mentionnée dans [CRM03] et calculée à partir du coefficient de Bhattacharyya pour mesurer la similarité entre deux densités de probabilités. Cette distance $d_B$ est calculée entre deux densités de probabilités $p$ et $q$ sur le support $X$ par :

$$\begin{aligned} d_B &= \sqrt{1 - \sum_{x \in X} \sqrt{p(x)q(x)}} \\ &= \sqrt{1 - BC} \end{aligned} \tag{6.28}$$

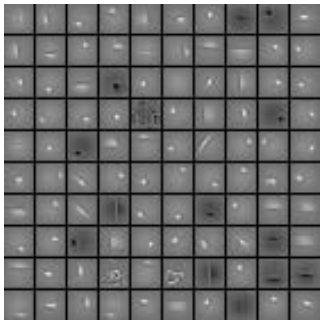
avec $BC$ le coefficient de Bhattacharyya. Pour deux densités de probabilités égales, $BC$ est égal à 1, rendant la distance $d_B$ nulle. $BC$ est au minimum égal à 0, la distance $d_B$ maximale étant alors égale à 1. Pour chaque pixel, quatre distances sont ainsi calculées par classe $c$ : une par composante couleur ($d_{BRc}$, $d_{BVc}$, $d_{BBc}$ et $d_{BIRc}$).

Le terme d'attache aux données $Data_{cost}$, constitué d'une valeur de pénalité par pixel et par classe, donné aux algorithmes de lissage, devient alors pour un pixel $pix$ et la classe $c$ :

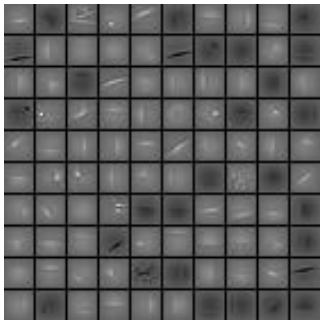
$$Data_{cost}(pix, c) = Err(y, D_c) + \gamma \frac{(d_{BRc} + d_{BVc} + d_{BBc} + d_{BIRc})}{4} \tag{6.29}$$



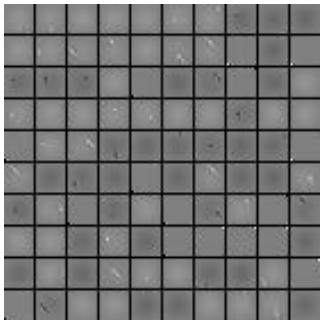
(a) Désert



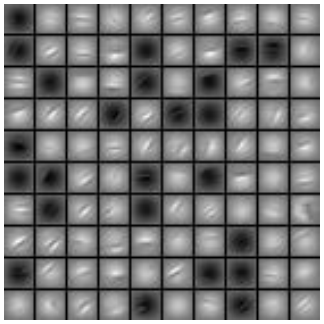
(b) Ville



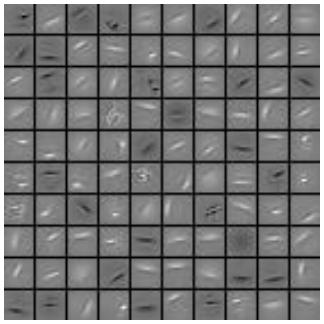
(c) Campagne



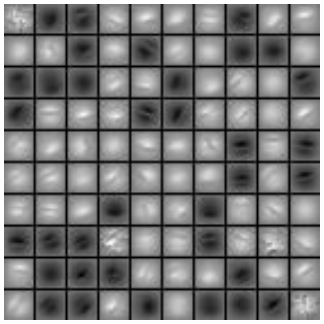
(d) Mer



(e) Neige



(f) Montagne



(g) Nuage

FIGURE 6.10 – Dictionnaires au premier niveau des Structures Adaptatives apprises pour les différentes classes des images satellites (dictionnaires 169x100). Les atomes de chaque dictionnaire sont rangés en colonnes. Chaque atome, rangé en colonnes, est représenté sous la forme d'un bloc.

le premier terme étant lié à l'erreur de reconstruction du patch $y$, centré sur le pixel $pix$, sur le dictionnaire $D_c$ de la classe $c$, et le second terme correspondant au terme de pénalité lié à la couleur, calculé par une moyenne des distances sur chaque composante couleur entre la densité de probabilité du patch $y$ et la densité de probabilité calculée sur les données d'apprentissage de la classe $c$. Le paramètre $\gamma$ permet de pondérer les deux termes de pénalité, sa valeur étant empiriquement fixée à 0.003.

**Résultats**

Nous ne disposons pas pour cette collection d'images satellites d'une vérité terrain permettant de calculer un pourcentage de mauvaises classifications, nous nous contenterons donc de juger visuellement de la qualité de classification des différentes images. Cependant, nous disposons tout de même de méta-données indiquant les classes présentes dans chaque image, sans toutefois connaître avec précision la classe associée à chaque pixel. A noter que ces méta-données n'ont pas été utilisées pour la classification. Contrairement aux expérimentations sur la base de texture, on considère qu'on ne sait pas quelles classes sont présentes dans l'image de test et chaque patch est donc décomposé sur l'ensemble des dictionnaires.

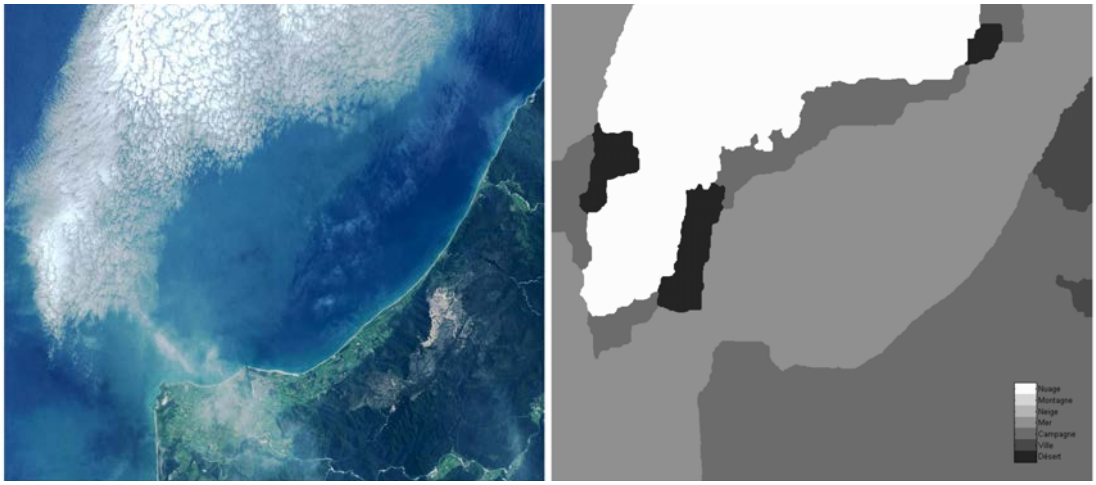
Pour l'image 174 (Fig. 6.11), les trois classes majoritaires sont "campagne", "mer" et "nuage". On retrouve donc bien les méta-données fournies. L'emplacement des labels est globalement correct. Des effets de bords sur le nuages donnent cependant les labels "campagne" et "désert" à certains pixels. On retrouve également quelques zones de label "ville" au sein du label "campagne".

Sur l'image 184 (Fig. 6.12), la mer est bien classifiée. Sur la zone terrestre en haut à droite de l'image, on retrouve la ville, entourée de campagne. Les méta-données sont donc retrouvées. Cependant, le label "neige" apparaît également en gris clair en haut de cette zone ainsi que sur l'île de gauche (en plus du label "campagne"). A noter que la petite île au centre peut également être retrouvée (en tant que "campagne") si les paramètres d'érosion sont adaptés pour ne pas éroder cette petite zone.

Quant à l'image 195 (Fig. 6.13), on y retrouve bien les zones de nuages correctement segmentées. Le reste de l'image, représentant de la campagne, est majoritairement bien classifié (les classes majoritaires correspondent donc bien aux méta-données "campagne" et "nuage"), hormis certaines zones de "ville" en gris plus foncé.

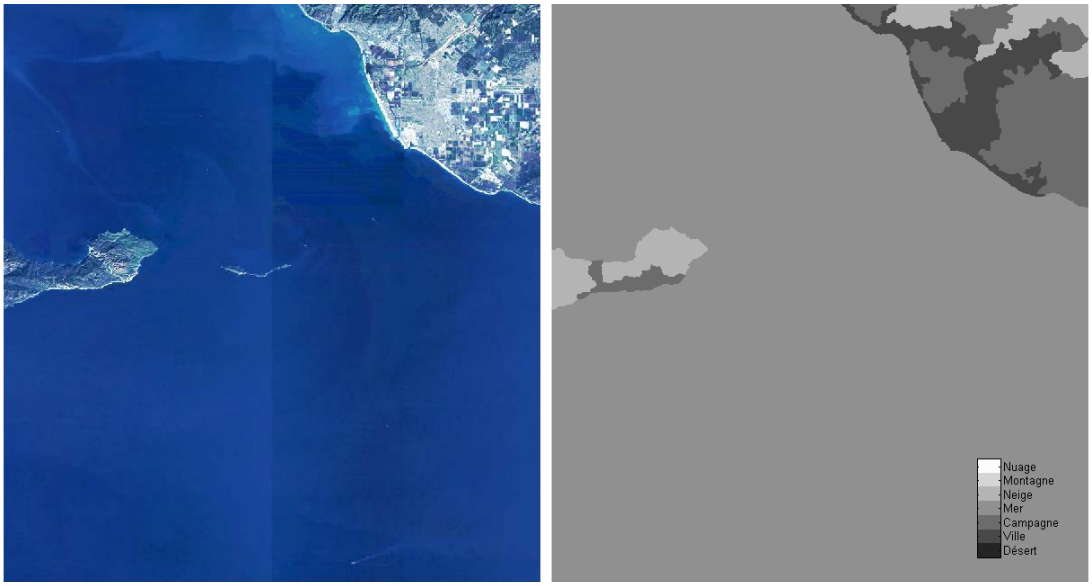
Les images suivantes sont davantage problématiques. Sur l'image 132 (Fig. 6.14) la zone de mer est globalement bien classifiée. Mais le désert est majoritairement classifié comme de la neige (en gris clair), sauf au bas de l'image où les pixels sont bien labellisés "désert" (en noir). La ville, ainsi que la côte, sont classifiées en tant que campagne. Seul un petit label de ville (en gris foncé) apparaît au milieu de la ville.

Sur l'image 116 (Fig. 6.15), la zone de campagne à droite est correctement classifiée, ainsi que la petite ville sur la droite. Quelques zones sont correctement classifiées comme des nuages au bas de l'image (blanc). Cependant, le désert et la montagne présents sur l'image sont majoritairement classifiés comme de la neige (gris clair), ainsi que comme de la campagne (gris foncé) et du désert (noir). Encore une fois, on retrouve donc une confusion avec les labels de "neige" et de "campagne".



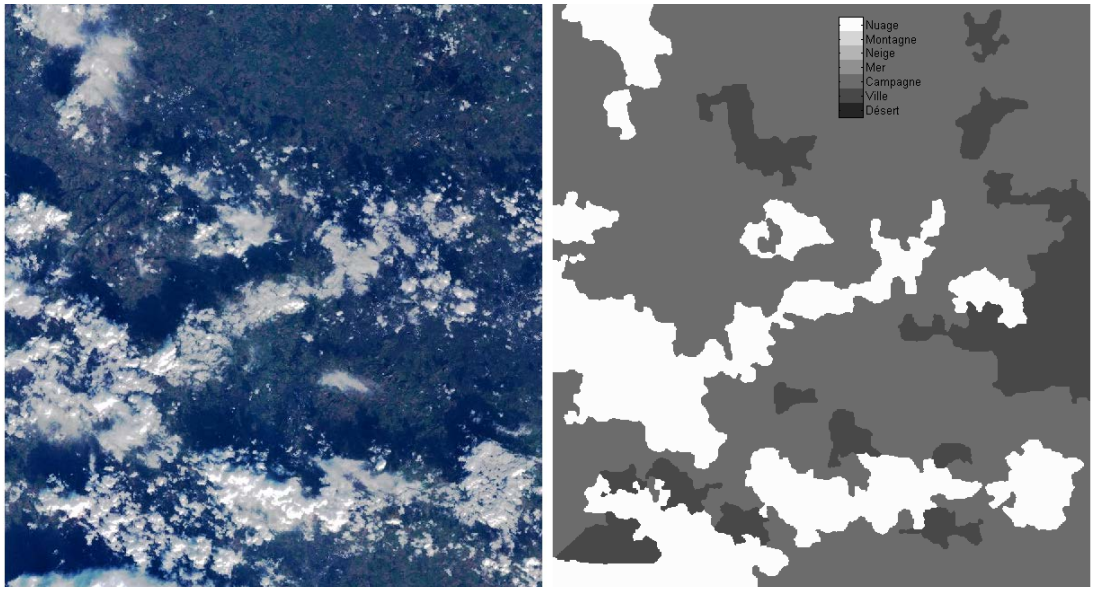

(a) Image satellite 174 (b) Image de labels

FIGURE 6.11 – Image satellite 174 et son image de labels après classification ($\gamma = 0.003$). Méta-données : campagne, mer, nuage. Pourcentages des classes : mer-41%, campagne-33%, nuage-21%,désert-3%, ville-2%, neige-0%, montagne-0%.




(a) Image satellite 184 (b) Image de labels

FIGURE 6.12 – Image satellite 184 et son image de labels après classification ($\gamma = 0.003$). Méta-données : ville, campagne, mer. Pourcentages des classes : mer-88%, campagne-6%, neige-3%, ville-3%, désert-0%, nuage-0%, montagne-0%.




(a) Image satellite 195 (b) Image de labels

FIGURE 6.13 – Image satellite 195 et son image de labels après classification ($\gamma = 0.003$). Méta-données : campagne, nuage. Pourcentages des classes : campagne-66%, nuage-22%, ville-12%, mer-0%, neige-0%, désert-0%, montagne-0%.

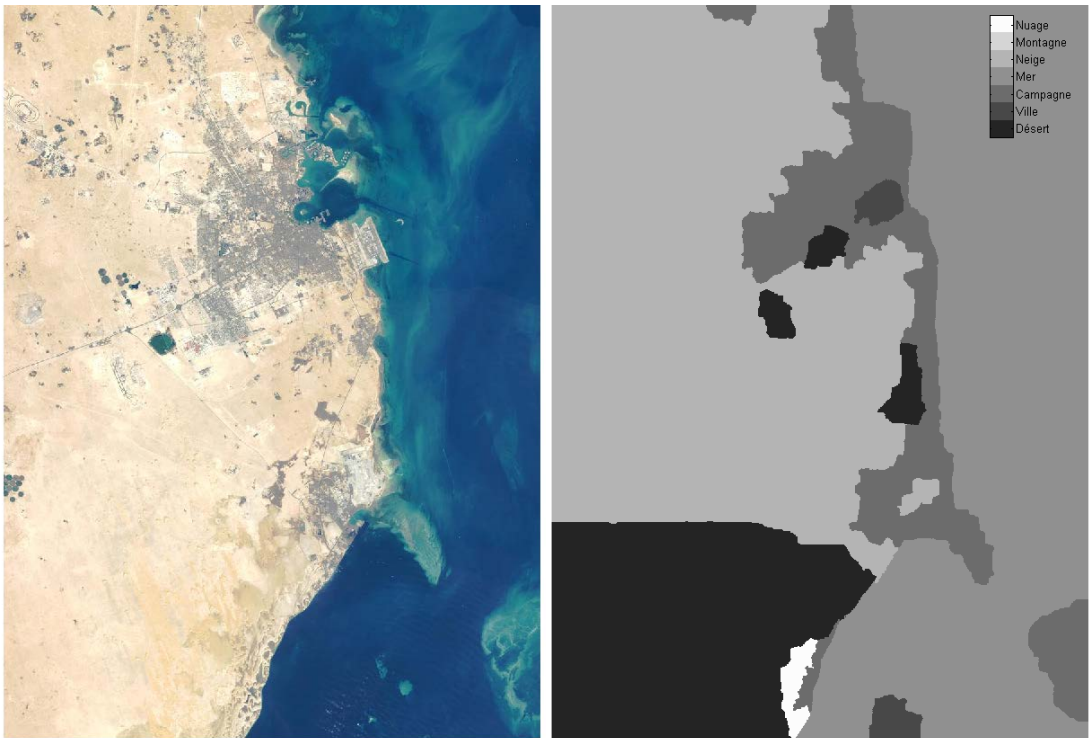
L'image 190 (Fig. 6.16) est sur sa partie de droite majoritairement classifiée comme de la montagne (gris claire) et de la neige (gris). La montagne enneigée est globalement classifiée en montagne de part sa structure, et non par sa couleur, qui la classifierait plutôt comme de la neige, voire des nuages. On retrouve d'ailleurs quelques zones classifiées à tort en nuages (blanc). La partie gauche de l'image est quant à elle classifiée comme de la campagne et de la ville, au lieu de la montagne.

Enfin, sur l'image 181 (Fig. 6.17), la zone de nuage est bien classifiée comme telle (blanc). En revanche, la mer, en général bien classifiée, est ici labellisée en tant que campagne. La mer présente sur l'image est sombre par rapport aux patchs de mer sélectionnés sur les images d'apprentissage. Cela peut expliquer l'erreur de classification. Des patchs de mer plus sombre dans la base d'apprentissage de la classe "mer" corrigeraient probablement le problème. Encore une fois, les zones de désert et de montagne sont majoritairement classifiées comme de la neige, malgré une zone de désert (noir) bien classifiée en haut à droite de l'image.

A la vue des différentes images de label obtenues, le principal problème semble être la confusion vers le label de "neige", en particulier du label de "désert". Structurellement, il est en effet difficile de les discriminer grâce aux dictionnaires. Leurs dictionnaires au premier niveau de leur Structure Adaptative respective présentent d'ailleurs des atomes relativement proches (Fig. 6.10). De plus, la neige peut sur certains patchs présenter des

structures liées à la classe présente sous la neige, ce qui peut entraîner des confusions de certains labels vers le label de "neige". Il serait donc nécessaire d'accorder pour ces zones davantage de poids au terme de pénalité lié à la couleur, c'est à dire augmenter la valeur du $\gamma$.

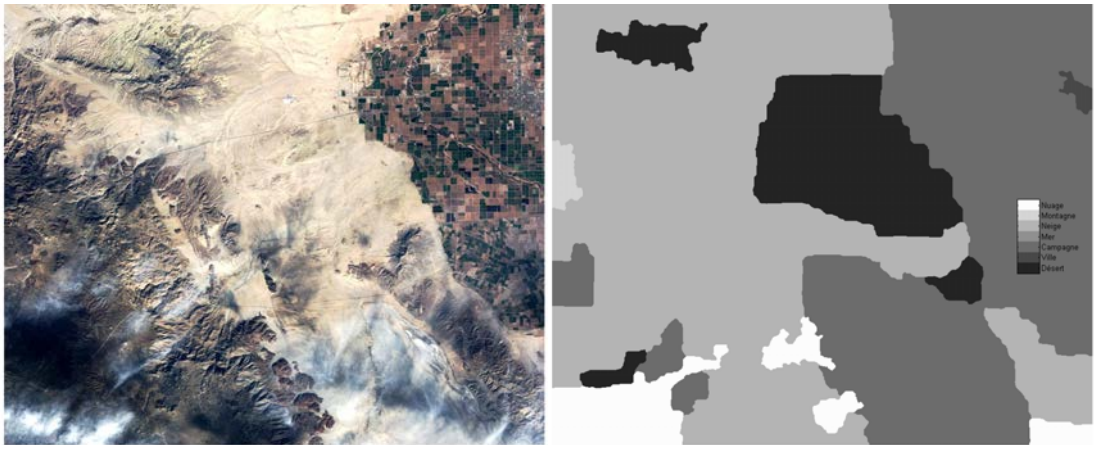
Un autre problème est la confusion entre les classes "montagne", "campagne" et "ville". Pour la classe "ville", il est possible que nous soyons pénalisés par le faible nombre d'exemples de villes dans la base d'apprentissage. En effet, les patchs d'entraînement extraits pour cette classe proviennent majoritairement d'une seule image de ville, ce qui ne permet pas d'avoir un apprentissage riche. Quant à la classe "campagne", elle peut regrouper des patchs très variés et son dictionnaire peut ainsi être capable de représenter des patchs d'autres classes, d'où le nombre de confusions importantes vers la classe "campagne". Les atomes de son dictionnaire au premier niveau sont d'ailleurs très diversifiés (Fig. 6.10) et présentent des similitudes notamment avec certains atomes des dictionnaires des labels "ville" et "montagne".




(a) Image satellite 132 (b) Image de labels

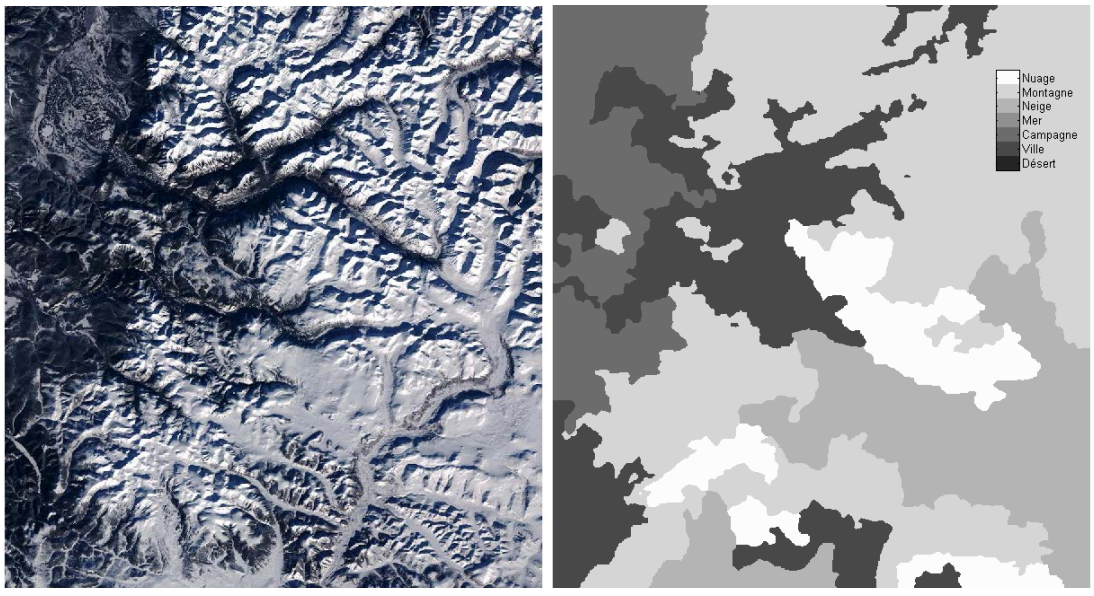
FIGURE 6.14 – Image satellite 132 et son image de labels après classification ($\gamma = 0.003$). Méta-données : désert, mer, ville. Pourcentages des classes : neige-40%, mer-33%, désert-16%, campagne-10%, ville-1%, nuage-0%, montagne-0%.

A présent, essayons d'accorder davantage de poids au terme de pénalité sur la




(a) Image satellite 116 (b) Image de labels

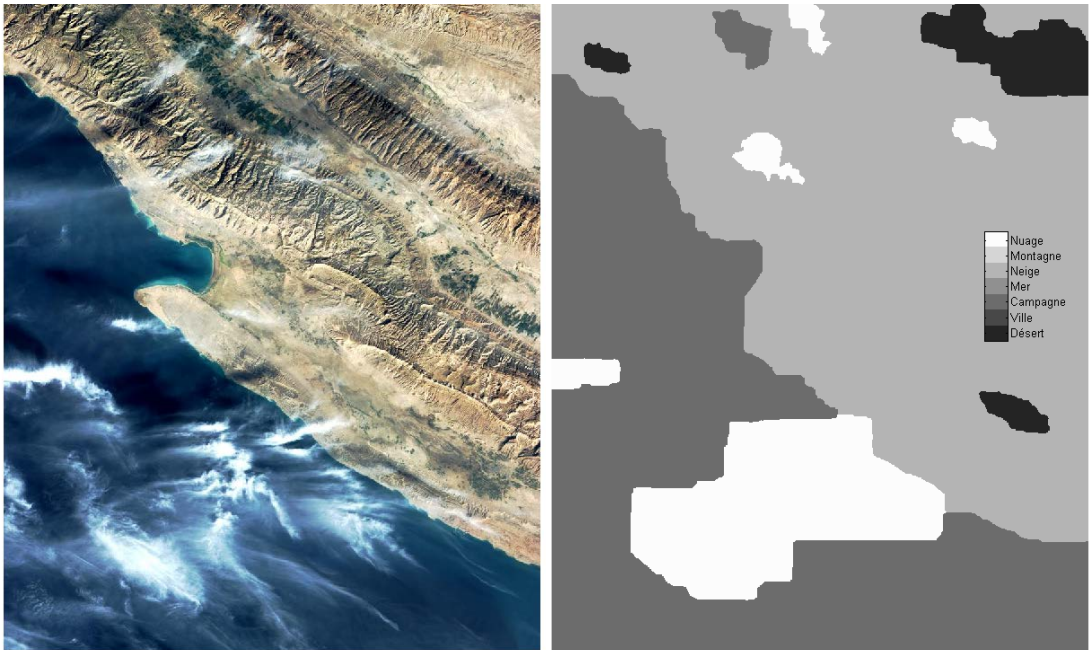
FIGURE 6.15 – Image satellite 116 et son image de labels après classification ($\gamma = 0.003$). Méta-données : campagne, désert, montagne. Pourcentages des classes : neige-45%, campagne-36%, désert-12%, nuage-5%, montagne-1%, ville-0%, mer-0%.




(a) Image satellite 190 (b) Image de labels

FIGURE 6.16 – Image satellite 190 et son image de labels après classification ($\gamma = 0.003$). Méta-données : neige, montagne. Pourcentages des classes : montagne-42%, ville-20%, neige-17%, campagne-12%, nuage-9%, désert-0%, mer-0%.

couleur, en augmentant la valeur du $\gamma$ à 0.015, afin de mieux discerner la neige des autres classes.




(a) Image satellite 181 (b) Image de labels

FIGURE 6.17 – Image satellite 181 et son image de labels après classification ($\gamma = 0.003$). Méta-données : mer, désert, montagne. Pourcentages des classes : neige-49%, campagne-36%, nuage-12%, désert-3%, mer-0%, ville-0%, montagne-0%.

Sur l'image 132 (Fig. 6.18), la zone de désert est grâce à l'augmentation du $\gamma$ majoritairement classifiée comme du désert, malgré quelques zones labellisées comme de la neige restantes.

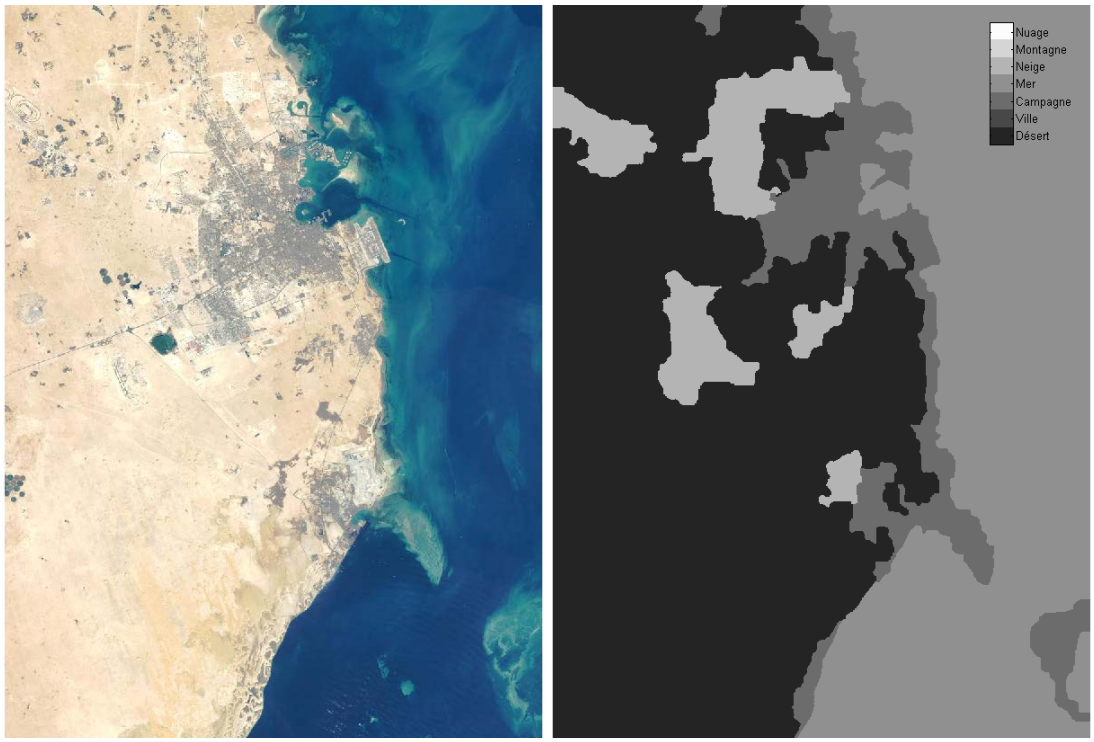
Sur l'image 116 (Fig. 6.19), le label "neige" a disparu au profit des labels "campagne" et "désert" qui se sont étendus. Des zones de montagnes sont mêmes classifiées comme telles (gris clair). Mais le label "campagne" est trop présent, au détriment des labels "désert" et "montagne". Les trois classes dominantes correspondent cependant bien aux méta-données.

Quant à l'image 181 (Fig. 6.20), le label de "neige" a principalement été remplacé par le label de "désert", ainsi que par les labels de "campagne" et de "nuages", même si quelques zones de neige perdurent encore. Sur la zone de mer, on retrouve le label de "ville" sur certaines zones, en plus du label de "campagne" majoritaire, probablement à cause de la couleur sombre de la mer.

Enfin, sur l'image 184 (Fig. 6.21), les quelques petites zones labellisées comme de la neige ont été remplacées par des zones de montagne (gris clair). La label "ville" (gris foncé) s'est de plus étendu sur le label "campagne" (gris).

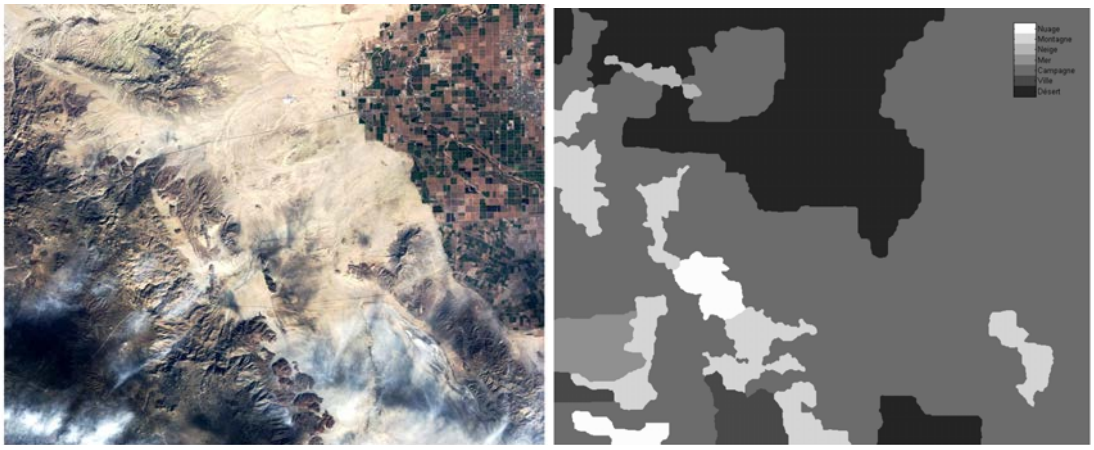
Le fait d'accorder plus de poids à la couleur a ainsi permis de mieux discriminer la neige des autres classes, en particulier du désert. Cependant, cette valeur de $\gamma$ dégrade la classification d'autres images telles que la 174 et la 195, où des confusions entre

classes apparaissent.



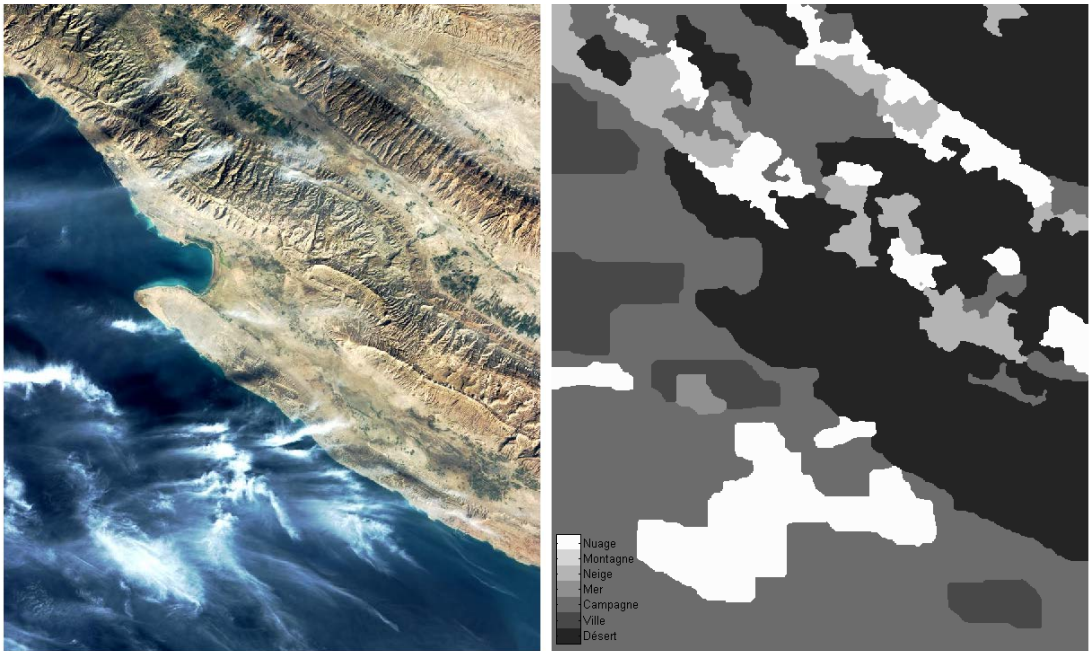

(a) Image satellite 132 (b) Image de labels

FIGURE 6.18 – Image satellite 132 et son image de labels après classification ($\gamma = 0.015$). Méta-données : désert, mer, ville. Pourcentages des classes : désert-49%, mer-35%, campagne-9%, neige-7%, ville-0%, nuage-0%, montagne-0%.



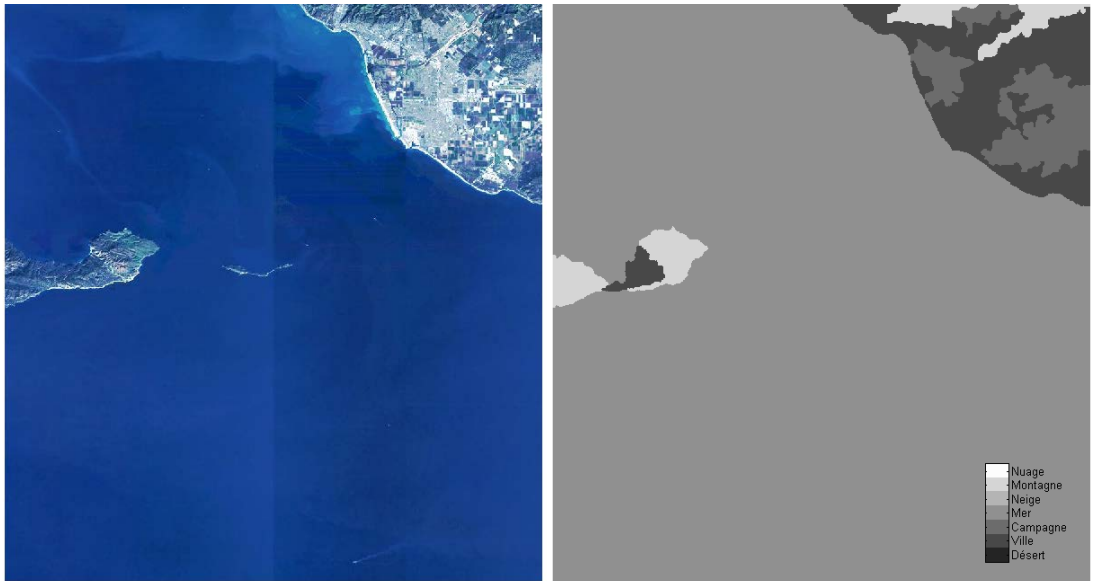

(a) Image satellite 116 (b) Image de labels

FIGURE 6.19 – Image satellite 116 et son image de labels après classification ($\gamma = 0.015$). Méta-données : campagne, désert, montagne. Pourcentages des classes : campagne-64%, désert-22%, montagne-8%, ville-2%, mer-2%, nuage-2%, neige-0%.




(a) Image satellite 181 (b) Image de labels

FIGURE 6.20 – Image satellite 181 et son image de labels après classification ($\gamma = 0.015$). Méta-données : mer, désert, montagne. Pourcentages des classes : campagne-40%, désert-35%, nuage-13%, ville-6%, neige-6%, mer-0%, montagne-0%.




(a) Image satellite 184 (b) Image de labels

FIGURE 6.21 – Image satellite 184 et son image de labels après classification ($\gamma = 0.015$). Méta-données : ville, campagne, mer. Pourcentages des classes : mer-88%, ville-6%, campagne-3%, montagne-3%, neige-0%, désert-0%, nuage-0%.

## 6.5 Conclusion

Dans ce chapitre, nous avons étudié les performances des dictionnaires structurés appris dans le cadre de la reconnaissance de scènes. L'objectif est de retrouver au sein d'une image inconnue des types de scènes connus tels que de la ville, ou encore des nuages. Cela correspond ainsi à une problème de classification supervisée.

Afin de traiter ce problème, les dictionnaires ont été adaptés en les discriminant lors de l'apprentissage de façon à ce qu'ils soient bons pour une classe et mauvais pour les autres. Une matrice d'affinité permet de plus ou moins discriminer une classe par rapport à chacune des autres classes selon leurs affinités. Un dictionnaire est appris par classe et un patch de classe inconnue est classifié d'après les erreurs de reconstruction obtenues sur les différents dictionnaires. Des étapes de lissage permettent ensuite d'obtenir une segmentation propre de l'image test en différentes classes.

Tout d'abord appliqué à une base connue de douze images de texture, notre algorithme a permis d'obtenir des résultats à la hauteur de l'état de l'art avec un pourcentage de mauvaises classifications moyen de 2.86%.

Nous l'avons ensuite testé sur des images satellites afin de classifier ces images en sept classes : désert, ville, campagne, mer, neige, montagne et nuage. Par rapport aux images de texture précédentes, le problème est plus complexe, certaines classes ne présentant pas de structures particulières sur lesquelles les discriminer. Les informations de couleur ont alors été ajoutées afin d'aider la classification, sans nécessiter de ré-

apprentissage des dictionnaires. Les résultats obtenus sont encourageants sur certaines images, mais des confusions entre classes sont encore présentes.

La méthode requiert le réglage de nombreux paramètres et il est donc difficile de faire varier l'ensemble des paramètres. C'est pourquoi certains paramètres ont été fixés tandis que d'autres ont été déterminés de manière empirique. Il est probablement possible, en faisant varier certains paramètres, d'améliorer encore les résultats obtenus, aussi bien sur les images de texture que les images satellites. De plus, certains paramètres déterminés sur les images de texture ont été utilisés identiques pour la classification des images satellites. On pense en particulier au paramètre $\alpha$ (intervenant dans le calcul du $\lambda$) réglant la pondération entre les termes de reconstruction et de discrimination au premier niveau des structures. Mais les images de texture et les images satellites étant de nature différente, il aurait pu être bénéfique d'ajuster ce paramètre $\alpha$ pour la classification des images satellites.

On pourrait également envisager d'autres méthodes de calcul de la matrice d'affinité (par exemple via une PCA (*Principal Component Analysis*) réalisée sur les données de chaque classe), utilisée pour la discrimination des dictionnaires à l'apprentissage, et en étudier l'impact sur les résultats. Ensuite, bien que cela risque d'augmenter la complexité de l'apprentissage, il serait intéressant de discriminer les structures de dictionnaires sur tous les niveaux et non simplement au premier. D'autres modèles de surexposition pourraient peut-être également aider à améliorer la classification de l'image test de texture n°6 en particulier.

Enfin, la classification sur les images satellites avec les classes précédemment citées fait apparaître un cas d'étude non présent pour les images de textures : le problème des zones pouvant présenter plusieurs labels. En effet, nous avons étudié un algorithme affectant un unique label à chaque pixel, mais certaines zones pourraient être définies par plusieurs labels, telles que les zones de montagne enneigée (montagne et neige), de ville enneigée (ville et neige) ou encore de campagne nuageuse (campagne et nuage). Actuellement, ces zones peuvent poser des problèmes à l'algorithme de classification. Il serait ainsi intéressant de poursuivre cette étude en incorporant la possibilité d'affecter plusieurs labels à un pixel, ou bien de créer des classes intermédiaires, en apprenant par exemple un dictionnaire spécialement dédié à une classe “montagne enneigée”, à mi-chemin entre les classes “montagne” et “neige”.

# Conclusion

Dans le cadre de cette thèse, nous avons proposé de travailler sur les méthodes de représentations parcimonieuses et d'apprentissage de dictionnaires utilisées pour deux finalités applicatives : la compression et la classification d'images satellites. Le principe est ainsi d'adapter des dictionnaires à un type d'images particulières, les images satellites. Les dictionnaires sont structurés afin dans un premier temps d'être plus adaptés au problème de codage. Leur capacité d'apprentissage est par la suite utilisée pour les problèmes d'estimation de FTM et de reconnaissance de scènes.

## Résumé des contributions de la thèse

Nous avons dans un premier temps travaillé sur différentes structures de dictionnaires. Ces structures sont composées de multiples dictionnaires organisés en niveaux, chaque dictionnaire à un niveau donné étant appris sur des résidus du niveau supérieur, par l'algorithme K-SVD pour une parcimonie de 1 atome. A la décomposition, un atome est sélectionné par niveau. Ces structures ont pour but d'offrir un meilleur compromis entre erreur de représentation et coût de codage que les dictionnaires "plats" tels que K-SVD. Elles permettent en effet de contenir de nombreux dictionnaires et donc davantage d'atomes pour améliorer la représentation, tout en conservant un coût de codage lié aux indices codés individuellement et une complexité de décomposition équivalents de part leur structuration en petits dictionnaires. Elles sont de plus scalables en parcimonie puisque non apprises pour une valeur de parcimonie précise. Nous avons débuté avec une structure arborescente, mais le nombre de dictionnaires devenant rapidement trop important en augmentant le nombre de niveaux, une structure en "cerf-volant" a été proposée, permettant d'élaguer les branches à un niveau donné pour n'apprendre ensuite qu'un seul dictionnaire par niveau. Le paramètre du niveau de fermeture des branches optimal dépendant de l'image considérée, la Structure Adaptative a été créée. Cette structure s'adapte aux données d'apprentissage en refermant progressivement ses différentes branches pour les fusionner au sein d'une branche commune. Cette structure offre à parcimonie égale une meilleure qualité de reconstruction qu'un dictionnaire K-SVD.

Un schéma de codage existant développé autour des dictionnaires structurés a ensuite été adapté pour la Structure Adaptative et quelques améliorations y ont été apportées. Le codeur fonctionne pour un débit cible et sélectionne à chaque étape un bloc

pour lequel ajouter un atome dans sa représentation, selon un critère débit-distorsion. Le codage des valeurs moyennes des blocs ainsi que l'apprentissage des tables de Huffman pour le codage des coefficients ont été améliorés, et la quantification a été intégrée dans la décomposition. Comparé à des codeurs standards, ce schéma utilisé avec la Structure Adaptative permet d'atteindre de meilleurs résultats que JPEG et est capable de surpasser CCSDS 122 sur une certaine plage de débits (typiquement jusqu'à 0.6 bpp). Autour de 0.3-0.4 bpp, il peut selon les images s'approcher voire atteindre les résultats de JPEG 2000. Le codeur peut ainsi sur ces débits rivaliser avec des codeurs standards optimisés, avec un codage entropique simple comparé à celui de JPEG 2000.

Les dictionnaires structurés appris ont ensuite été utilisés dans un algorithme d'estimation de FTM instrument à partir d'images satellites. Contrairement aux méthodes usuelles d'estimations, cette méthode ne nécessite pas l'utilisation d'une mire ou la recherche de forts contours dans l'image. Elle est basée sur l'apprentissage d'un dictionnaire structuré pour chaque valeur de FTM. La décomposition de l'image test de FTM inconnue sur les différents dictionnaires permet, grâce à un algorithme de décision utilisant des seuils préalablement déterminés, d'estimer la valeur de FTM liée à l'image. Les résultats obtenus permettent d'estimer la FTM avec une erreur moyenne faible, correspondant visuellement à un écart de FTM quasiment imperceptible.

Enfin, les dictionnaires ont été discriminés et intégrés dans un algorithme de classification supervisée dans le cadre de la reconnaissance de scènes au sein d'une image satellite. Un dictionnaire structuré discriminant est appris par classe, la discrimination ayant pour but de rendre les dictionnaires bons pour leur classe mais également mauvais pour les autres classes. Un patch de classe inconnue est alors classifié d'après les erreurs de reconstruction calculées sur les différents dictionnaires. Des méthodes de lissage permettent ensuite d'obtenir une segmentation propre de l'image en plusieurs labels. D'abord testé sur une base connue d'images de textures, l'algorithme a permis d'obtenir des résultats à la hauteur des derniers résultats de la littérature sur cette base d'images. L'algorithme a ensuite été appliqué à des images satellites afin d'y retrouver par exemple des scènes de mer, de ville ou encore de désert. Une pénalité liée à la couleur a également été ajoutée pour aider la classification. Les résultats obtenus sont satisfaisants sur certaines images mais présentent encore des confusions entre classes sur d'autres, certaines classes étant difficiles à discriminer.

## Perspectives

Plusieurs perspectives peuvent être envisagées afin de poursuivre les travaux réalisés dans cette thèse.

Tout d'abord, au sein des structures que nous avons développées (Chapitre 3), il serait envisageable d'avoir un nombre d'atomes par dictionnaire variable, nombre qui pourrait dépendre du niveau du dictionnaire dans la structure, ou bien du nombre de vecteurs d'entraînement disponibles pour apprendre ce dictionnaire.

Ensuite, d'autres critères de fermeture des branches pourraient être testés au sein de la Structure Adaptative. Par exemple, au lieu de refermer les branches d'après le nombre de vecteurs d'apprentissage dans la branche, on pourrait envisager un critère basé sur le contenu de ces vecteurs d'apprentissage en cherchant à déterminer si ce contenu nécessite un dictionnaire spécifiquement appris sur ces vecteurs.

Un autre axe de travail serait de chercher à réduire la complexité des décompositions réalisées sur les dictionnaires structurés, en particulier si celles-ci sont effectuées à bord d'un satellite. On pourrait par exemple pour la recherche de l'atome le plus corrélé au résidu courant calculer la corrélation sur un sous-ensemble des deux vecteurs afin de limiter le nombre de multiplications réalisées, au prix d'une recherche moins optimale.

A plus long terme, il serait intéressant d'étudier des méthodes d'apprentissage profond, les réseaux de neurones étant comme les dictionnaires structurés organisés en plusieurs couches successives. Comme au sein des structures de dictionnaires que nous avons présentées, l'apprentissage est dans certaines études réalisé couche par couche, la représentation apprise pour une couche étant utilisée comme entrée de la suivante. Un traitement non-linaire est de plus associé à chaque nœud du réseau de neurones. Afin de limiter les paramètres à estimer, des réseaux de neurones convolutifs peuvent être utilisés, les paramètres du filtre de convolution, appliqué à un support plus grand, sont alors partagés entre plusieurs branches. Ces concepts d'apprentissage profond, appliqués à l'apprentissage des dictionnaires structurés, pourraient ainsi permettre d'améliorer leur efficacité.

En ce qui concerne le codage (Chapitre 4), nous avons vu que l'apprentissage est particulièrement efficace lorsque les données de test et d'apprentissage sont très proches, comme dans le cas de l'observation persistante. Ainsi, cela pourrait potentiellement s'étendre au cas des satellites géostationnaires ciblant toujours la même zone de la Terre. Les premières images capturées par le satellite serviraient d'images d'apprentissage d'un dictionnaire structuré. L'apprentissage serait alors réalisé sur Terre, ce qui nécessiterait de transmettre le dictionnaire, ou à bord du satellite, ce qui risquerait d'être assez lourd en calculs. Puis ce dictionnaire serait utilisé pour coder les images suivantes, avec un recalage effectué avec les images d'apprentissage pour une meilleure efficacité.

Une autre perspective serait d'apprendre un dictionnaire encore davantage adapté au problème du codage. Au niveau du débit, les données les plus lourdes à transmettre sont les indices des atomes sélectionnés pour l'approximation des signaux. Malheureusement, ces indices présentent une forte entropie du fait de leur distribution quasiment uniforme. On pourrait alors penser à apprendre un dictionnaire dont les atomes sont de différentes importances, par exemple selon leur utilisation par les données d'apprentissage. Lors de la décomposition, il serait alors possible de favoriser les atomes les plus populaires. Plutôt que de sélectionner uniquement l'atome le plus corrélé, on pourrait sélectionner les $k$ ($k \in \mathbb{N}$) atomes les plus corrélés (ou utiliser un seuil sur le niveau de corrélation pour sélectionner plusieurs atomes) et choisir parmi ces atomes le plus populaire dans le dictionnaire. Bien que cela induise forcément une perte en qualité de reconstruction, cela permettra de déséquilibrer l'utilisation des atomes du dictionnaires en favorisant les atomes les plus populaires, de façon à diminuer l'entropie des indices

pour par exemple les coder, comme pour les coefficients, avec des tables de Huffman, en accordant ainsi des codes de plus petites tailles aux indices des atomes les plus populaires dans le dictionnaire. Il faudrait ensuite étudier si un tel dictionnaire peut permettre d'améliorer le compromis débit-distorsion.

Enfin, le schéma pourrait s'adapter aux données locales de l'image en travaillant avec des blocs de différentes tailles. A la manière du quadtree de HEVC, la taille des blocs pourrait être adaptée en fonction du contenu local de l'image, c'est-à-dire dans notre cas d'après la taille des structures à apprendre ou l'homogénéité du bloc.

Concernant la méthode d'estimation de FTM (Chapitre 5), celle-ci fait intervenir une fonction affine de seuils. Il serait intéressant de tester des fonctions de seuils plus complexes et non linéaires.

De plus, les résultats présentés au Chapitre 5 ne considèrent l'utilisation que d'une seule image test pour réaliser l'estimation. Ainsi, si on se place dans un cas où il est possible d'obtenir plusieurs images issues du même capteur, alors l'estimation peut être réalisée sur plus de données et ainsi être affinée. Il serait donc intéressant de réaliser cette estimation de FTM dans ce cas plus favorable et observer les potentiels gains obtenus.

Pour l'algorithme de classification supervisée utilisé dans le cadre de la reconnaissance de scènes (Chapitre 6), l'impact de la matrice d'affinité sur la discrimination mériterait d'être davantage étudié. D'autres méthodes de calcul de cette matrice pourraient être envisagées, par exemple en réalisant une PCA sur les données de chaque classe pour trouver les vecteurs caractéristiques des classes. Il serait de plus intéressant de discriminer les structures de dictionnaires sur tous les niveaux, au lieu de se limiter à la discrimination du premier niveau. Concernant le modèle de sur-exposition, d'autres modèles plus complexes pourraient être expérimentés.

On peut également envisager d'ajouter d'autres critères permettant d'améliorer la segmentation de l'image, tels que des informations de contours. On pourrait ainsi limiter le lissage d'une zone, en particulier une zone homogène, une fois ses contours déterminés.

De plus, afin de classifier correctement des zones ambigües comme de la montagne enneigée ou de la campagne nuageuse, l'algorithme, attribuant un unique label a chaque pixel, pourrait être étendu avec la possibilité d'affecter plusieurs labels à un pixel. Une solution serait l'apprentissage de dictionnaires intermédiaires, à mi-chemin entre deux classes. On apprendrait par exemple un dictionnaire spécial pour les zones de montagne enneigée. Une autre solution serait de réaliser la classification en plusieurs étapes. Plutôt que de choisir une classe parmi toutes celles disponibles, on effectuerait d'abord une pré-classification par exemple pour les classes de nuage ou de neige entre "présence" ou "absence", avec ensuite la possibilité d'ajouter un label lié aux autres classes.

Pour aller plus loin, la méthode de classification pourrait évoluer vers une méthode de classification hiérarchique réalisée en plusieurs étapes, chaque étape étant associée à un critère de classification particulier. On pourrait par exemple envisager une première étape classifiant les patchs sur un critère structurel, puis une seconde utilisant un critère calorimétrique.

A propos de la couleur, cette dernière pourrait être intégrée dès l'apprentissage, en cherchant à appendre des dictionnaires discriminants selon les structures et les couleurs. Mais cela augmenterait également la quantité de données à traiter et conduirait donc à une hausse de la complexité d'apprentissage des dictionnaires et de décompositions des patchs de test.

Enfin, la classification actuelle pourrait être remplacée par une classification non déterministe. Pour chaque pixel, plutôt que de lui affecter un unique label, une mesure de confiance pourrait être calculée par classe, basée sur l'erreur de reconstruction calculée sur chacun des dictionnaires.

# Glossaire

AVC : Advanced Video Coding
BITD : Basic Iteration-Tuned Dictionary
BP : Basis Pursuit
bpp : bit(s) par pixel
CCSDS : Consultative Comittee for Space Data Systems
DCT : Discrete Cosine Transform
DFT : Discrete Fourier Transform
DPCM : Differential Pulse-Code Modulation
DWT : Discrete Wavelet Transform
EBCOT : Embedded Block Coding with Optimized Truncation
EOB : End Of Block
ESF : Edge Spread Function
FTM : Fonction de Transfert de Modulation
GSD : Ground Sample Distance
HEVC : High Efficiency Video Coding
ITAD : Iteration-Tuned and Aligned Dictionary
ITD : Iteration-Tuned Dictionary
JPEG : Joint Photographic Experts Group
LSF : Line Spread Function
MOD : Method of Optimal Directions
MP : Matching Pursuit
NMF : Non-negative Matrix Factorization
OMP : Orthogonal Matching Pursuit
PSF : Point Spread Function
PSNR : Peak Signal-to-Noise Ratio
RMSE : Root Mean Square Error
SAAN : Sélection Adaptative des Atomes par Niveau
SAD : Sum of Absolute Differences
SVD : Singular Value Decomposition
TSITD : Tree-Structured Iteration-Tuned Dictionary

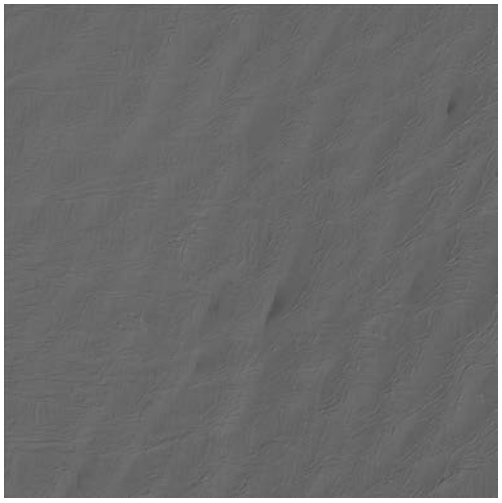
# Annexe 1

(a) Boston18

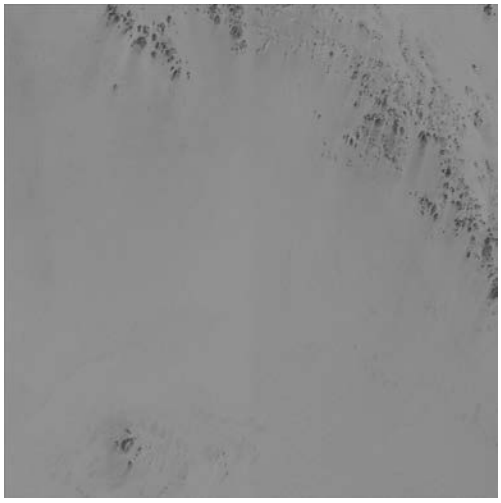
(b) Champagne18

(c) Correze20

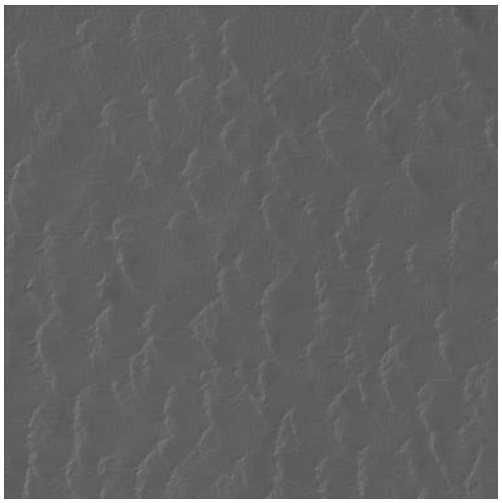
(d) Dubai18



(e) Libye4

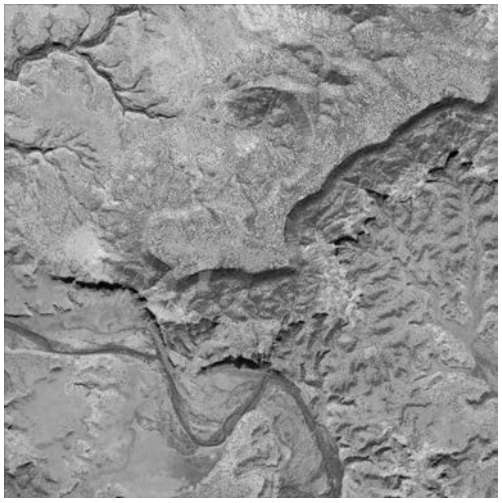


(f) Libye8

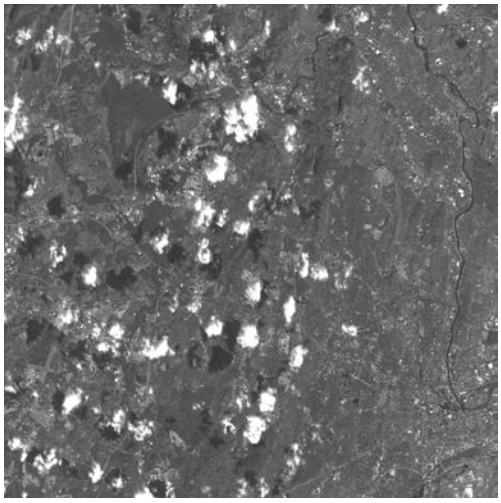


(g) Libye12

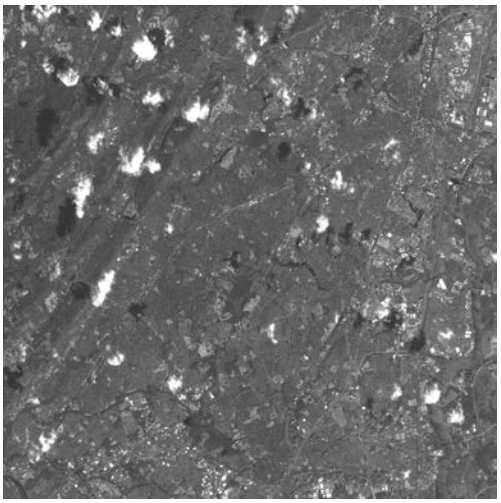
(h) Mataro19



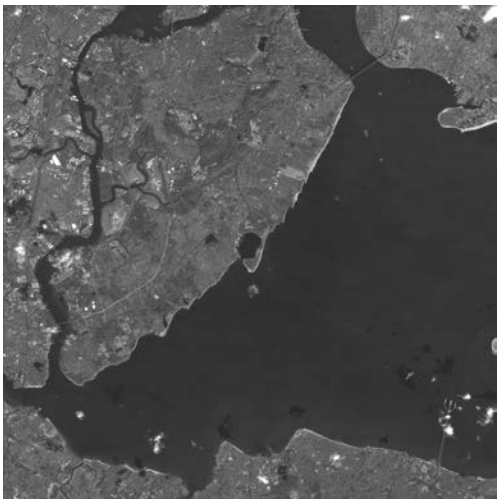
(i) MontSaintMichel18



(j) NewYork1



(k) NewYork2

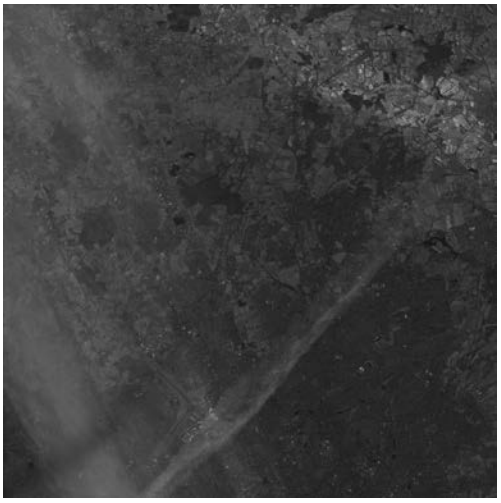


(l) NewYork4

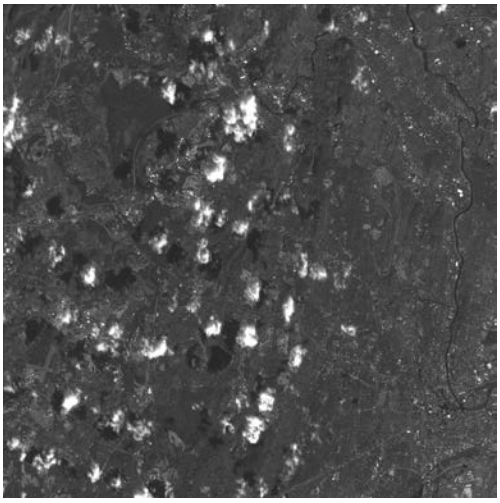
(m) Sochaux19

FIGURE 22 – Base d'images d'apprentissage utilisées dans la partie 2 de cette thèse (FTM de valeur 0.35)

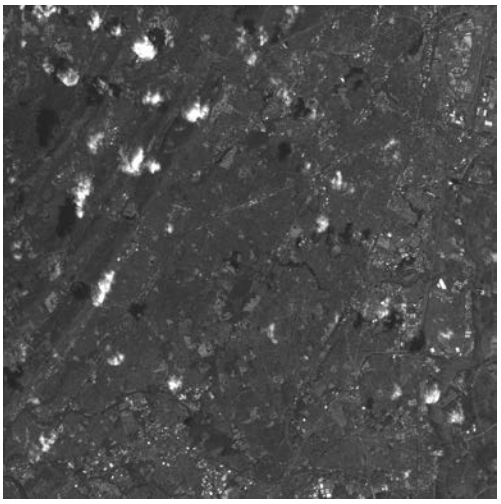
# Annexe 2



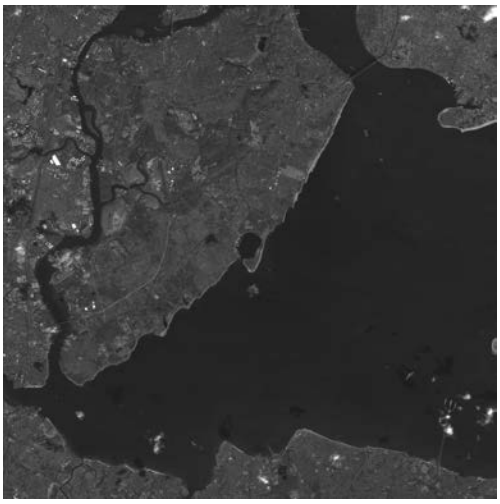
(a) Hambourg2



(b) NewYork1



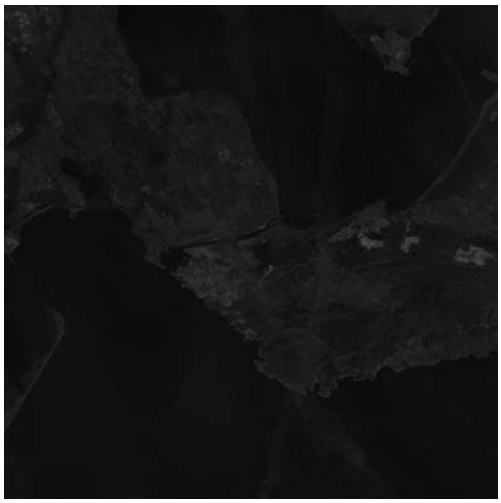
(c) NewYork2



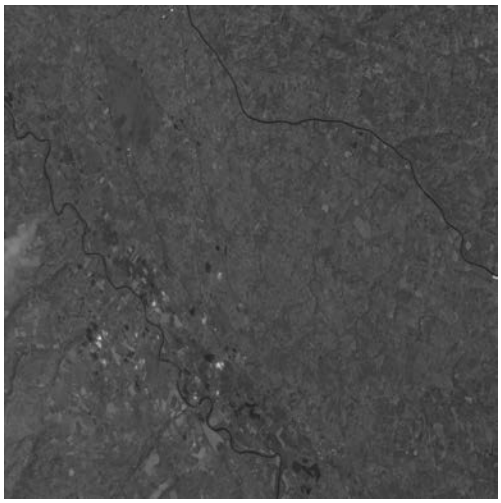
(d) NewYork4

(e) SalonProvence1

(f) SalonProvence2

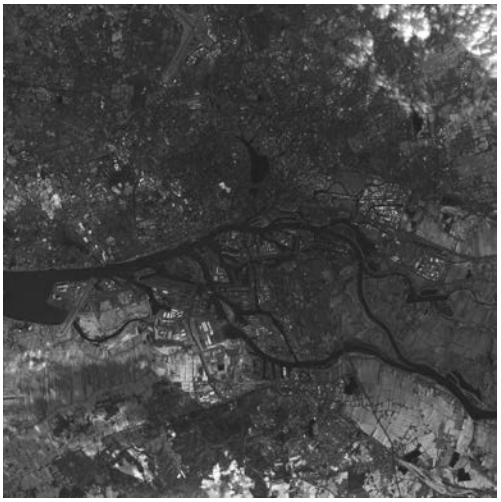


(g) SalonProvence4

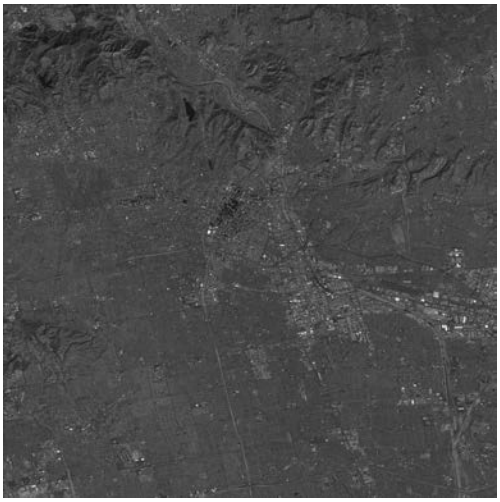


(h) Toulouse1

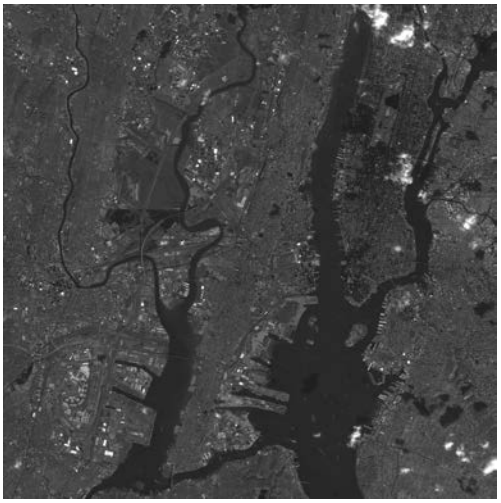
FIGURE 23 – Base d'images d'apprentissage utilisées dans le chapitre 5 (Estimation de FTM) (paragraphe 5.4 Expérimentations) de cette thèse (FTM de valeur 0.40).



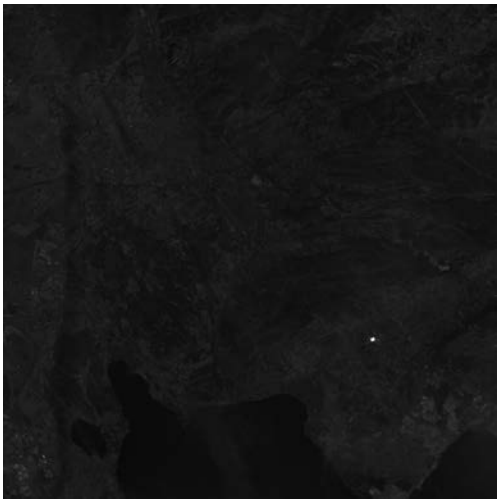
(a) Hambourg1



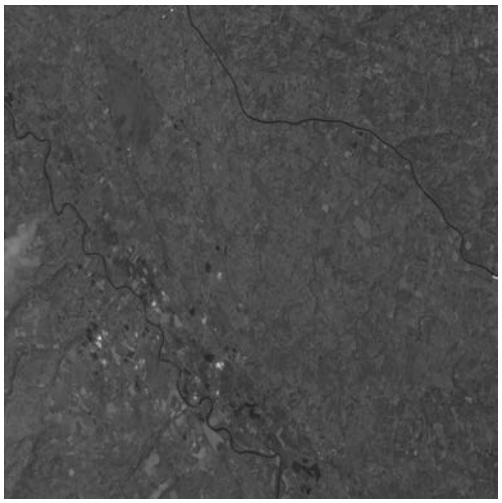
(b) LosAngeles1



(c) NewYork3



(d) SalonProvence3



(e) Toulouse1

FIGURE 24 – Images de tests utilisées dans le chapitre 5 (Estimation de FTM) (paragraphe 5.4 Expérimentations) de cette thèse (FTM de valeur 0.40)

# Bibliographie


[ABMD92] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies. Image coding using wavelet transform. *IEEE Transactions on Image Processing*, 1(2) :205–220, 1992.

[AE08] M. Aharon and M. Elad. Sparse and redundant modeling of image content using an image-signature-dictionary. *SIAM Journal on Imaging Sciences*, 1(3) :228–247, 2008.

[AEB] M. Aharon, M. Elad, and A. Bruckstein. K-SVD Matlab toolbox. *Available at : http ://www.cs.technion.ac.il/∼elad/software/.*

[AEB05] M. Aharon, M. Elad, and A. M. Bruckstein. K-SVD and its non-negative variant for dictionary design. In *Optics & Photonics 2005*. International Society for Optics and Photonics, 2005.

[AEB06] M. Aharon, M. Elad, and A. Bruckstein. K-SVD : An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation. *IEEE Transactions on Signal Processing*, 54(11) :4311–4322, 2006.

[AMGL13a] J. Aghaei Mazaheri, C. Guillemot, and C. Labit. Learning a tree-structured dictionary for efficient image representation with adaptive sparse coding. In *ICASSP*, 2013.

[AMGL13b] J. Aghaei Mazaheri, C. Guillemot, and C. Labit. Learning an adaptive dictionary structure for efficient image sparse coding. In *PCS-30th Picture Coding Symposium-2013*, 2013.

[AMGL13c] J. Aghaei Mazaheri, C. Guillemot, and C. Labit. Représentations parcimonieuses par un dictionnaire à structure adaptative : application au codage d'images satellitaires. In *24ème colloque Gretsi*, 2013.

[ANR74] N. Ahmed, T. Natarajan, and K. R. Rao. Discrete cosine transform. *IEEE Transactions on Computers*, 100(1) :90–93, 1974.

[Bag06] S. Bagon. Matlab Wrapper for Graph Cut, December 2006. *Available at : http ://www.wisdom.weizmann.ac.il/∼bagon/matlab.html.*

[BD08] T. Blumensath and M. E. Davies. Iterative thresholding for sparse approximations. *Journal of Fourier Analysis and Applications*, 14(5-6) :629–654, 2008.

[BE08] O. Bryt and M. Elad. Compression of facial images using the K-SVD algorithm. *Journal of Visual Communication and Image Representation*, 19(4) :270–282, 2008.

[Ber99] D. P. Bertsekas. *Nonlinear programming*. 1999.

[BK04] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9) :1124–1137, 2004.

[BLF13] G. Blanchet, C. Latry, and S. Fourest. Mesures automatiques des performances radiométriques et géométriques des satellites Pléiades. Détail de la mesure de la FTM. In *Gretsi*, September 2013.

[Bro66] P. Brodatz. *Textures : A Photographic Album for Artists and Designers*. New york : Dover edition, 1966.

[BVZ01] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11) :1222–1239, 2001.

[CCS] CCSDS. CCSDS - Implementation Software & Guides. *Available at : http ://public.ccsds.org/implementations/software.aspx*.

[CDF92] A. Cohen, I. Daubechies, and J.-C. Feauveau. Biorthogonal bases of compactly supported wavelets. *Communications on pure and applied mathematics*, 45(5) :485–560, 1992.

[CDS98] S. S. Chen, D. L. Donoho, and M. A. Saunders. Atomic decomposition by basis pursuit. *SIAM journal on scientific computing*, 20(1) :33–61, 1998.

[CFJ08] V. Cheung, B. J. Frey, and N. Jojic. Video epitomes. *International Journal of Computer Vision*, 76(2) :141–152, 2008.

[CGT+11] S. Cherigui, C. Guillemot, D. Thoreau, P. Guillotel, and P. Perez. Epitome-based image compression using translational sub-pel mapping. In *IEEE 13th International Workshop on Multimedia Signal Processing (MMSP), 2011*, pages 1–6. IEEE, 2011.

[CR01] S. F. Cotter and B. D. Rao. Application of tree-based searches to matching pursuit. In *IEEE International Conference on Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01).*, volume 6, pages 3933–3936. IEEE, 2001.

[CRM03] D. Comaniciu, V. Ramesh, and P. Meer. Kernel-based object tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5) :564–577, 2003.

[Dau06] L. Daudet. Sparse and structured decompositions of signals with the molecular matching pursuit. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(5) :1808–1816, 2006.

[DFKE07] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian. Image denoising by sparse 3-D transform-domain collaborative filtering. *IEEE Transactions on Image Processing*, 16(8) :2080–2095, 2007.

[DLMS07] A. Di Lillo, G. Motta, and J. A. Storer. Texture classification based on discriminative features extracted in the frequency domain. In *IEEE International Conference on Image Processing (ICIP)*, volume 2, pages II–53, 2007.

[DLRV03] J.-M. Delvit, D. Leger, S. Roques, and C. Valorge. Estimation de la fonction de transfert de modulation à l'aide d'un réseau de neurones. In *Gretsi*, 2003.

[DLZS11] W. Dong, X. Li, D. Zhang, and G. Shi. Sparsity-based image denoising via dictionary learning and structural clustering. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 457–464. IEEE, 2011.

[DTDS12] D. L. Donoho, Y. Tsaig, I. Drori, and J.-L. Starck. Sparse solution of underdetermined systems of linear equations by stagewise orthogonal matching pursuit. *IEEE Transactions on Information Theory*, 58(2) :1094–1121, 2012.

[EA06a] M. Elad and M. Aharon. Image denoising via learned dictionaries and sparse representation. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 895–900. IEEE, 2006.

[EA06b] M. Elad and M. Aharon. Image denoising via sparse and redundant representations over learned dictionaries. *IEEE Transactions on Image Processing*, 15(12) :3736–3745, 2006.

[EAH99] K. Engan, S. O. Aase, and J. H. Husoy. Method of optimal directions for frame design. In *IEEE International Conference on Acoustics, Speech, and Signal Processing, 1999*, volume 5, pages 2443–2446. IEEE, 1999.

[EAH00] K. Engan, S. O. Aase, and J. H. Husøy. Multi-frame compression : Theory and design. *Signal Processing*, 80(10) :2121–2140, 2000.

[EHJ+04] B. Efron, T. Hastie, I. Johnstone, R. Tibshirani, and others. Least angle regression. *The Annals of statistics*, 32(2) :407–499, 2004.

[ER06] R. Eslami and H. Radha. Translation-invariant contourlet transform and its application to image denoising. *IEEE Transactions on Image Processing*, 15(11) :3362–3374, 2006.

[ERKD99] K. Engan, B. D. Rao, and K. Kreutz-Delgado. Frame design using FOCUSS with method of optimal directions (MOD). In *Proc. NORSIG*, volume 99, 1999.

[FEYM09] S. Faran, I. Eshet, N. Yehezkel, and J. Molcho. Estimation of the MTF of a satellite imaging-system from celestial scenes. In *2009 IEEE International Geoscience and Remote Sensing Symposium, IGARSS 2009*, volume 2, pages II–452. IEEE, 2009.

[FiVVF06] R. M. Figueras i Ventura, P. Vandergheynst, and P. Frossard. Low-rate and flexible image coding with redundant representations. *IEEE Transactions on Image Processing*, 15(3) :726–739, 2006.

[fSDS05] The Consultative Committee for Space Data Systems. Recommendation for Space Data System Standard. Image Data Compression. Recommended standard CCSDS 122.0-B-1, 2005.

[GG92] A. Gersho and R. M. Gray. *Vector quantization and signal compression*. Springer, 1992.

[GGK11] M. J. Gangeh, A. Ghodsi, and M. S. Kamel. Dictionary learning in texture classification. In *Image Analysis and Recognition*, pages 335–343. Springer, 2011.

[HBR12] I. Horev, O. Bryt, and R. Rubinstein. Adaptive image compression using sparse dictionaries. In *19th International Conference on Systems, Signals and Image Processing (IWSSIP)*, pages 592–595. IEEE, 2012.

[Hoy02] P. O. Hoyer. Non-negative sparse coding. In *Proceedings of the 2002 12th IEEE Workshop on Neural Networks for Signal Processing, 2002.*, pages 557–565. IEEE, 2002.

[Hoy04] P. O. Hoyer. Non-negative matrix factorization with sparseness constraints. *The Journal of Machine Learning Research*, 5 :1457–1469, 2004.

[Huf52] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9) :1098–1101, 1952.

[JFK03] N. Jojic, B. J. Frey, and A. Kannan. Epitomic analysis of appearance and shape. In *Proceedings. Ninth IEEE International Conference on Computer Vision, 2003.*, pages 34–41, 2003.

[JLD11] Z. Jiang, Z. Lin, and L. S. Davis. Learning a discriminative dictionary for sparse coding via label consistent K-SVD. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1697–1704. IEEE, 2011.

[JMOB10a] R. Jenatton, J. Mairal, G. Obozinski, and F. Bach. Proximal methods for hierarchical sparse coding. *Arxiv preprint arXiv :1009.2139*, 2010.

[JMOB10b] R. Jenatton, J. Mairal, G. Obozinski, and F. Bach. Proximal methods for sparse hierarchical dictionary learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2010.

[JVF06] P. Jost, P. Vandergheynst, and P. Frossard. Tree-based pursuit : Algorithm and properties. *IEEE Transactions on Signal Processing*, 54(12) :4685–4697, 2006.

[KL80] V. Klema and A. J. Laub. The singular value decomposition : Its computation and some applications. *IEEE Transactions on Automatic Control*, 25(2) :164–176, 1980.

[KMS+12] I.-K. Kim, K. McCann, K. Sugimoto, B. Bross, and W.-J. Han. HM9 : High Efficiency Video Coding (HEVC) Test Model 9 Encoder Description, 2012.

[Koh04] K. Kohm. Modulation transfer function measurement method and results for the Orbview-3 high resolution imaging satellite. In *Proceedings of ISPRS*, pages 12–23, 2004.

[KW12] S. Kong and D. Wang. A dictionary learning approach for classification : separating the particularity and the commonality. In *Computer Vision–ECCV 2012*, pages 186–199. Springer, 2012.

[KZ04] V. Kolmogorov and R. Zabin. What energy functions can be minimized via graph cuts ? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2) :147–159, 2004.

[LBH+12] J. Lainema, F. Bossen, W.-J. Han, J. Min, and K. Ugur. Intra coding of the HEVC standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12) :1792–1801, 2012.

[LD06] C. La and M. N. Do. Tree-based orthogonal matching pursuit algorithm for signal reconstruction. In *IEEE International Conference on Image Processing, 2006*, pages 1277–1280. IEEE, 2006.

[LDR94] D. Leger, J. Duffaut, and F. Robinet. MTF measurement using spotlight. In *International Geoscience and Remote Sensing Symposium, 1994. IGARSS'94.*, volume 4, pages 2010–2012. IEEE, 1994.

[LGT88] D. Le Gall and A. Tabatabai. Sub-band coding of digital images using symmetric short kernel filters and arithmetic coding techniques. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 761–764. IEEE, 1988.

[LS00] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In *Advances in neural information processing systems*, pages 556–562, 2000.

[MBP+08a] J. Mairal, F. Bach, J. Ponce, G. Sapiro, and A. Zisserman. Discriminative learned dictionaries for local image analysis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8. IEEE, 2008.

[MBP+08b] J. Mairal, F. Bach, J. Ponce, G. Sapiro, and A. Zisserman. Supervised dictionary learning. In *Advances in neural information processing systems (NIPS 2008)*, pages 1033–1040, 2008.

[MBPS10] J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online learning for matrix factorization and sparse coding. *The Journal of Machine Learning Research*, 11 :19–60, 2010.

[MEZ05] B. Matalon, M. Elad, and M. Zibulevsky. Improved denoising of images using modelling of a redundant contourlet transform. In *Optics & Photonics 2005*, pages 59141Y–59141Y. International Society for Optics and Photonics, 2005.

[MGBB00] M. W. Marcellin, M. J. Gormish, A. Bilgin, and M. P. Boliek. An overview of JPEG-2000. In *Data Compression Conference (DCC)*, pages 523–541. IEEE, 2000.

[MJV04] G. Monaci, P. Jost, and P. Vandergheynst. Image compression with learnt tree-structured dictionaries. In *IEEE 6th Workshop on Multimedia Signal Processing*, pages 35–38, 2004.

[MPO00] T. Mäenpää, M. Pietikäinen, and T. Ojala. Texture classification by multi-predicate local binary pattern operators. In *Proc. ICPR*, 2000.

[MSW03] D. Marpe, H. Schwarz, and T. Wiegand. Context-based adaptive binary arithmetic coding in the H. 264/AVC video compression standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7) :620–636, 2003.

[MV04] G. Monaci and P. Vandergheynst. Learning structured dictionaries for image representation. In *International Conference on Image Processing (ICIP)*, volume 4, pages 2351–2354. IEEE, 2004.

[MZ93] S. G. Mallat and Z. Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, 41(12) :3397–3415, 1993.

[NB01] N. R. Nelson and P. S. Barry. Measurement of Hyperion MTF from on-orbit scenes. In *IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, volume 7, pages 2967–2969. IEEE, 2001.

[NNI09] M. Nakashizuka, H. Nishiura, and Y. Iiguni. Sparse image representations with shift-invariant tree-structured dictionaries. In *16th IEEE International Conference on Image Processing (ICIP)*, pages 2145 –2148, November 2009.

[PE09] M. Protter and M. Elad. Image sequence denoising via sparse and redundant representations. *IEEE Transactions on Image Processing*, 18(1) :27–35, 2009.

[Pot52] R. B. Potts. Some generalized order-disorder transformations. In *Mathematical proceedings of the cambridge philosophical society*, volume 48, pages 106–109. Cambridge Univ Press, 1952.

[PRK93] Y. C. Pati, R. Rezaiifar, and P. S. Krishnaprasad. Orthogonal matching pursuit : Recursive function approximation with applications to wavelet decomposition. In *Conference Record of The Twenty-Seventh Asilomar Conference on Signals, Systems and Computers.*, pages 40–44, 1993.

[PSWS03] J. Portilla, V. Strela, M. J. Wainwright, and E. P. Simoncelli. Image denoising using scale mixtures of Gaussians in the wavelet domain. *IEEE Transactions on Image Processing*, 12(11) :1338–1351, 2003.

[Ran] T. Randen. Trygve Randen web page. *Available at : http ://www.ux.uis.no/∼tranden/.*

[RG08] G. Rath and C. Guillemot. A complementary matching pursuit algorithm for sparse approximation. In *Proc. European Signal Processing Conference (EUSIPCO)*, volume 164, 2008.

[RH99] T. Randen and J. H. Husoy. Filtering for texture classification : A comparative study. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(4) :291–310, 1999.

[RNL02] L. Rebollo-Neira and D. Lowe. Optimized orthogonal matching pursuit approach. *IEEE Signal Processing Letters*, 9(4) :137–140, 2002.

[RS08] F. Rodriguez and G. Sapiro. Sparse representations for image classification : Learning discriminative and reconstructive non-parametric dictionaries. Technical report, DTIC Document, 2008.

[RSS10] I. Ramirez, P. Sprechmann, and G. Sapiro. Classification and clustering via dictionary learning with structured incoherence and shared features. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3501–3508. IEEE, 2010.

[Rub] R. Rubinstein. Sparse K-SVD Matlab toolbox. *Available at : http ://www.cs.technion.ac.il/∼ronrubin/software.html*.

[RZE10] R. Rubinstein, M. Zibulevsky, and M. Elad. Double sparsity : Learning sparse dictionaries for sparse signal approximation. *IEEE Transactions on Signal Processing*, 58(3) :1553–1564, 2010.

[SB12] V. Sze and M. Budagavi. High throughput CABAC entropy coding in HEVC. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12) :1778–1791, 2012.

[SCD02] J.-L. Starck, E. J. Candès, and D. L. Donoho. The curvelet transform for image denoising. *IEEE Transactions on Image Processing*, 11(6) :670–684, 2002.

[SCE01] A. Skodras, C. Christopoulos, and T. Ebrahimi. The JPEG 2000 still image compression standard. *IEEE Signal Processing Magazine*, 18(5) :36–58, 2001.

[SE10] K. Skretting and K. Engan. Recursive least squares dictionary learning algorithm. *IEEE Transactions on Signal Processing*, 58(4) :2121–2130, 2010.

[SE11] K. Skretting and K. Engan. Image compression using learned dictionaries by RLS-DLA and compared with K-SVD. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1517–1520. IEEE, 2011.

[SE14] K. Skretting and K. Engan. Energy Minimization by alpha-Erosion for Supervised Texture Segmentation. In *11th International Conference on Image Analysis and Recognition (ICIAR)*, pages 207–214. Springer, 2014.

[SHG08] O. G. Sezer, O. Harmanci, and O. G. Guleryuz. Sparse orthonormal transforms for image compression. In *15th IEEE International Conference on Image Processing. ICIP 2008.*, pages 149–152. IEEE, 2008.

[SOHW12] G. J. Sullivan, J. Ohm, W.-J. Han, and T. Wiegand. Overview of the high efficiency video coding (HEVC) standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12) :1649–1668, 2012.

[Tau00] D. Taubman. High performance scalable image compression with EBCOT. *IEEE transactions on Image Processing*, 9(7) :1158–1170, 2000.

[TD] TU-Delft. Image and Video Compression Learning Tool Vc-Demo | Intelligent Systems Department. *Available at : http ://insy.ewi.tudelft.nl/content/image-and-video-compression-learning-tool-vcdemo.*

[TGS06] J. A. Tropp, A. C. Gilbert, and M. J. Strauss. Algorithms for simultaneous sparse approximation. Part I : Greedy pursuit. *Signal Processing*, 86(3) :572–588, 2006.

[TM02] D. S. Taubman and M. W. Marcellin. *JPEG2000 : Image compression fundamentals, standards and practice.* Kluwer Academic Publishers, 2002.

[UCL] UCL. OpenJPEG library : an open source JPEG 2000 codec. *Available at : http ://www.openjpeg.org/index.php ?menu=main.*

[Wal91] G. K. Wallace. The JPEG still picture compression standard. *Communications of the ACM*, 34(4) :30–44, 1991.

[Wal92] G. K. Wallace. The JPEG still picture compression standard. *IEEE Transactions on Consumer Electronics*, 38(1) :xviii–xxxiv, 1992.

[Wil98] D. Williams. What is an MTF ... and Why Should You Care ?, February 1998. RLG DigiNews, Volume 2, Number 1.

[WLL09] T. Wang, S. Li, and X. Li. An automatic MTF measurement method for remote sensing cameras. In *2nd IEEE International Conference on Computer Science and Information Technology, 2009. ICCSIT 2009.*, pages 245–248. IEEE, 2009.

[WSB] H. Wang, K. Sayood, and M. Bauer. CCSDS Image Data Compression Implementation (University of Nebraska). *Available at : http ://hyperspectral.unl.edu/.*

[WSBL03] T. Wiegand, G.J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7) :560 –576, July 2003.

[WWHG09] J. Wang, Q. Wan, A. Huang, and T. Gan. Tree-based multiscale pursuit. In *International Conference on Communications, Circuits and Systems, 2009. ICCCAS 2009.*, pages 521–524. IEEE, 2009.

[WWOH08] H. Wang, Y. Wexler, E. Ofek, and H. Hoppe. Factoring repeated content within and among images. In *ACM Transactions on Graphics (TOG)*, volume 27, page 14, 2008.

[WYG+09] J. Wright, A. Y. Yang, A. Ganesh, S. S. Sastry, and Y. Ma. Robust face recognition via sparse representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(2) :210–227, 2009.

[XLLZ14] M. Xu, S. Li, J. Lu, and W. Zhu. Compressibility Constrained Sparse Representation with Learnt Dictionary for Low Bit-rate Image Compression. *IEEE Transactions on Circuits and Systems for Video Technology*, 24(10) :1743–1757, 2014.

[XZ12] D. Xu and X. Zhang. Study of MTF measurement technique based on special object image analyzing. In *International Conference on Mechatronics and Automation (ICMA)*, pages 2109–2113. IEEE, 2012.

[YZF11] M. Yang, D. Zhang, and X. Feng. Fisher discrimination dictionary learning for sparse representation. In *IEEE International Conference on Computer Vision (ICCV)*, pages 543–550. IEEE, 2011.

[Zep10] J. Zepeda. *New sparse representation methods ; application to image compression and indexing*. PhD thesis, Université de Rennes 1, 2010.

[ZGK+10] J. Zepeda, C. Guillemot, E. Kijak, et al. The iteration-tuned dictionary for sparse representations. In *Proc. of the IEEE International Workshop on MMSP*, 2010.

[ZGK11] J. Zepeda, C. Guillemot, and E. Kijak. Image Compression Using Sparse Representations and the Iteration-Tuned and Aligned Dictionary. *IEEE Journal of Selected Topics in Signal Processing*, 5 :1061–1073, 2011.

[ZL10] Q. Zhang and B. Li. Discriminative K-SVD for dictionary learning in face recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2691–2698. IEEE, 2010.

[ZRY09] P. Zhao, G. Rocha, and B. Yu. The composite absolute penalties family for grouped and hierarchical variable selection. *The Annals of Statistics*, pages 3468–3497, 2009.

# Publications

J. Aghaei Mazaheri, C. Guillemot, and C. Labit. Learning a tree-structured dictionary for efficient image representation with adaptive sparse coding. In *International Conference on Acoustics, Speech and Signal Processing, ICASSP*, 2013.

J. Aghaei Mazaheri, C. Guillemot, and C. Labit. Représentations parcimonieuses par un dictionnaire à structure adaptative : application au codage d'images satellitaires. In *24ème colloque Gretsi*, 2013.

J. Aghaei Mazaheri, C. Guillemot, and C. Labit. Learning an adaptive dictionary structure for efficient image sparse coding. In *Picture Coding Symposium, PCS*, 2013.

# Table des figures

## Résumé

Cette thèse propose d'explorer des méthodes de représentations parcimonieuses et d'apprentissage de dictionnaires pour compresser et classifier des images satellites. Les représentations parcimonieuses consistent à approximer un signal par une combinaison linéaire de quelques colonnes, dites atomes, d'un dictionnaire, et ainsi à le représenter par seulement quelques coefficients non nuls contenus dans un vecteur parcimonieux. Afin d'améliorer la qualité des représentations et d'en augmenter la parcimonie, il est intéressant d'apprendre le dictionnaire. La première partie de la thèse présente un état de l'art consacré aux représentations parcimonieuses et aux méthodes d'apprentissage de dictionnaires. Diverses applications de ces méthodes y sont détaillées. Des standards de compression d'images sont également présentés. La deuxième partie traite de l'apprentissage de dictionnaires structurés sur plusieurs niveaux, d'une structure en arbre à une structure adaptative, et de leur application au cas de la compression d'images satellites en les intégrant dans un schéma de codage adapté. Enfin, la troisième partie est consacrée à l'utilisation des dictionnaires structurés appris pour la classification d'images satellites. Une méthode pour estimer la Fonction de Transfert de Modulation (FTM) de l'instrument dont provient une image est étudiée. Puis un algorithme de classification supervisée, utilisant des dictionnaires structurés rendus discriminants entre les classes à l'apprentissage, est présenté dans le cadre de la reconnaissance de scènes au sein d'une image.

## Abstract

This thesis explores sparse representation and dictionary learning methods to compress and classify satellite images. Sparse representations consist in approximating a signal by a linear combination of a few columns, known as atoms, from a dictionary, and thus representing it by only a few non-zero coefficients contained in a sparse vector. In order to improve the quality of the representations and to increase their sparsity, it is interesting to learn the dictionary. The first part of the thesis presents a state of the art about sparse representations and dictionary learning methods. Several applications of these methods are explored. Some image compression standards are also presented. The second part deals with the learning of dictionaries structured in several levels, from a tree structure to an adaptive structure, and their application to the compression of satellite images, by integrating them in an adapted coding scheme. Finally, the third part is about the use of learned structured dictionaries for the classification of satellite images. A method to estimate the Modulation Transfer Function (MTF) of the instrument used to capture an image is studied. A supervised classification algorithm, using structured dictionaries made discriminant between classes during the learning, is then presented in the scope of scene recognition in a picture.